Showing data from an API

Showing data from an API with custom styling and interaction

> Note: This example uses UIKit.

This example loads lighthouses in the United States from WikiData. It adds points to the map and applies dynamic styling to these points. When zooming in the dots become larger circles with a custom icon and the name of the lighthouse shown next to it. When tapping a callout is shown with the name of the lighthouse that was tapped on.

```
class WebAPIDataExample: UIViewController, MHMapViewDelegate {
    var mapView: MHMapView!

    override func viewDidLoad() {
        super.viewDidLoad()

        mapView = MHMapView(frame: view.bounds)
        mapView.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        mapView.setCenter(CLLocationCoordinate2D(latitude: 37.090240,
longitude: -95.712891), zoomLevel: 2, animated: false)
        mapView.delegate = self
        mapView.attributionButton.isHidden = true
        view.addSubview(mapView)

        // Add a single tap gesture recognizer. This gesture requires
the built-in MHMapView tap gestures (such as those for zoom and
annotation selection) to fail.
        let singleTap = UITapGestureRecognizer(target: self, action:
#selector(handleMapTap(sender:)))
        for recognizer in mapView.gestureRecognizers! where recognizer
is UITapGestureRecognizer {
            singleTap.require(toFail: recognizer)
        }
        mapView.addGestureRecognizer(singleTap)
    }

    func mapView(_: MHMapView, didFinishLoading _: MHStyle) {
        fetchPoints { [weak self] features in
            self?.addItemsToMap(features: features)
        }
    }

    func addItemsToMap(features: [MHPointFeatureClusterFeature]) {
        // MHMapView.style is optional, so you must guard against it not
being set.
        guard let style = mapView.style else { return }

        // You can add custom UIImages to the map style.
```

```
        // These can be referenced by an MHSymbolStyleLayer's iconImage
property.
        style.setImage(UIImage(named: "lighthouse")!, forName:
"lighthouse")

        // Add the features to the map as a shape source.
        let source = MHShapeSource(identifier: "us-lighthouses",
features: features, options: nil)
        style.addSource(source)

        let lighthouseColor = UIColor(red: 0.08, green: 0.44, blue:
0.96, alpha: 1.0)

        // Use MHCircleStyleLayer to represent the points with simple
circles.
        // In this case, we can use style functions to gradually change
properties between zoom level 2 and 7: the circle opacity from 50% to
100% and the circle radius from 2pt to 3pt.

        let circles = MHCircleStyleLayer(identifier: "lighthouse-
circles", source: source)
        circles.circleColor = NSExpression(forConstantValue:
lighthouseColor)

        // The circles should increase in opacity from 0.5 to 1 based on
zoom level.
        circles.circleOpacity = NSExpression(forMHInterpolating:
NSExpression.zoomLevelVariable, curveType:
MHExpressionInterpolationMode.linear, parameters: nil, stops:
NSExpression(forConstantValue: [2: 0.5, 7: 1]))
        circles.circleRadius = NSExpression(forMHInterpolating:
NSExpression.zoomLevelVariable, curveType:
MHExpressionInterpolationMode.linear, parameters: nil, stops:
NSExpression(forConstantValue: [2: 2, 7: 3]))

        // Use MHSymbolStyleLayer for more complex styling of points
including custom icons and text rendering.
        let symbols = MHSymbolStyleLayer(identifier: "lighthouse-
symbols", source: source)
        symbols.iconImageName = NSExpression(forConstantValue:
"lighthouse")
        symbols.iconColor = NSExpression(forConstantValue:
lighthouseColor)
        symbols.iconScale = NSExpression(forConstantValue: 0.5)
        symbols.iconHaloColor = NSExpression(forConstantValue:
UIColor.white.withAlphaComponent(0.5))
        symbols.iconHaloWidth = NSExpression(forConstantValue: 1)
        symbols.iconOpacity = NSExpression(forMHInterpolating:
NSExpression.zoomLevelVariable, curveType:
MHExpressionInterpolationMode.linear, parameters: nil, stops:
NSExpression(forConstantValue: [5.9: 0, 6: 1]))

        // "name" references the "name" key in an
```

```
MHPointFeatureClusterFeature's attributes dictionary.
        symbols.text = NSExpression(forKeyPath: "name")
        symbols.textColor = symbols.iconColor
        symbols.textFontSize = NSExpression(forMHInterpolating:
NSExpression.zoomLevelVariable, curveType:
MHExpressionInterpolationMode.linear, parameters: nil, stops:
NSExpression(forConstantValue: [10: 10, 16: 16]))
        symbols.textTranslation = NSExpression(forConstantValue:
NSValue(cgVector: CGVector(dx: 15, dy: 0)))
        symbols.textOpacity = symbols.iconOpacity
        symbols.textHaloColor = symbols.iconHaloColor
        symbols.textHaloWidth = symbols.iconHaloWidth
        symbols.textJustification = NSExpression(forConstantValue:
NSValue(mhTextJustification: .left))
        symbols.textAnchor = NSExpression(forConstantValue:
NSValue(mhTextAnchor: .left))

        style.addLayer(circles)
        style.addLayer(symbols)
    }

    // MARK: - Feature interaction

    @IBAction func handleMapTap(sender: UITapGestureRecognizer) {
        if sender.state == .ended {
            // Limit feature selection to just the following layer
identifiers.
            let layerIdentifiers: Set = ["lighthouse-symbols",
"lighthouse-circles"]

            // Try matching the exact point first.
            let point = sender.location(in: sender.view!)
            for feature in mapView.visibleFeatures(at: point,
styleLayerIdentifiers: layerIdentifiers)
                where feature is MHPointFeatureClusterFeature
            {
                guard let selectedFeature = feature as?
MHPointFeatureClusterFeature else {
                    fatalError("Failed to cast selected feature as
MHPointFeatureClusterFeature")
                }
                showCallout(feature: selectedFeature)
                return
            }

            let touchCoordinate = mapView.convert(point,
toCoordinateFrom: sender.view!)
            let touchLocation = CLLocation(latitude:
touchCoordinate.latitude, longitude: touchCoordinate.longitude)

            // Otherwise, get all features within a rect the size of a
touch (44x44).
            let touchRect = CGRect(origin: point, size:
```

```
.zero).insetBy(dx: -22.0, dy: -22.0)
            let possibleFeatures = mapView.visibleFeatures(in:
touchRect, styleLayerIdentifiers: Set(layerIdentifiers)).filter { $0 is
MHPointFeatureClusterFeature }

            // Select the closest feature to the touch center.
            let closestFeatures = possibleFeatures.sorted(by: {
                CLLocation(latitude: $0.coordinate.latitude, longitude:
$0.coordinate.longitude).distance(from: touchLocation) <
CLLocation(latitude: $1.coordinate.latitude, longitude:
$1.coordinate.longitude).distance(from: touchLocation)
            })
            if let feature = closestFeatures.first {
                guard let closestFeature = feature as?
MHPointFeatureClusterFeature else {
                    fatalError("Failed to cast selected feature as
MHPointFeatureClusterFeature")
                }
                showCallout(feature: closestFeature)
                return
            }

            // If no features were found, deselect the selected
annotation, if any.

mapView.deselectAnnotation(mapView.selectedAnnotations.first, animated:
true)
        }
    }

    func showCallout(feature: MHPointFeatureClusterFeature) {
        let point = MHPointFeatureClusterFeature()
        point.title = feature.attributes["name"] as? String
        point.coordinate = feature.coordinate

        // Selecting an feature that doesn't already exist on the map
will add a new annotation view.
        // We'll need to use the map's delegate methods to add an empty
annotation view and remove it when we're done selecting it.
        mapView.selectAnnotation(point, animated: true,
completionHandler: nil)
    }

    // MARK: - MHMapViewDelegate

    func mapView(_: MHMapView, annotationCanShowCallout _: MHAnnotation)
-> Bool {
        true
    }

    func mapView(_ mapView: MHMapView, didDeselect annotation:
MHAnnotation) {
        mapView.removeAnnotations([annotation])
```

```
    }

    func mapView(_: MHMapView, viewFor _: MHAnnotation) ->
MHAnnotationView? {
        // Create an empty view annotation. Set a frame to offset the
callout.
        MHAnnotationView(frame: CGRect(x: 0, y: 0, width: 20, height:
20))
    }

    // MARK: - Data fetching and parsing

    func fetchPoints(withCompletion completion: @escaping
(([MHPointFeatureClusterFeature]) -> Void)) {
        // Wikidata query for all lighthouses in the United States:
http://tinyurl.com/zrl2jc4
        let query = "SELECT DISTINCT ?item " +
            "?itemLabel ?coor ?image " +
            "WHERE " +
            "{ " +
            "?item wdt:P31 wd:Q39715 . " +
            "?item wdt:P17 wd:Q30 . " +
            "?item wdt:P625 ?coor . " +
            "OPTIONAL { ?item wdt:P18 ?image } . " +
            "SERVICE wikibase:label { bd:serviceParam wikibase:language
\"en\" } " +
            "} " +
            "ORDER BY ?itemLabel"

        let characterSet = NSMutableCharacterSet()
        characterSet.formUnion(with: CharacterSet.urlQueryAllowed)
        characterSet.removeCharacters(in: "?")
        characterSet.removeCharacters(in: "&")
        characterSet.removeCharacters(in: ":")

        let encodedQuery =
query.addingPercentEncoding(withAllowedCharacters: characterSet as
CharacterSet)!

        let request = URLRequest(url: URL(string:
"https://query.wikidata.org/sparql?query=\(encodedQuery)&format=json")!)

        URLSession.shared.dataTask(with: request, completionHandler: {
data, _, error in
            guard error == nil else {
                preconditionFailure("Failed to load GeoJSON data: \
(error!)")
            }

            guard
                let data,
                let json = try? JSONSerialization.jsonObject(with: data,
options: []) as? [String: AnyObject],
```

```
                let results = json["results"] as? [String: AnyObject],
                let items = results["bindings"] as? [[String:
AnyObject]]
            else {
                preconditionFailure("Failed to parse GeoJSON data")
            }

            DispatchQueue.main.async {
                completion(self.parseJSONItems(items: items))
            }
        }).resume()
    }

    func parseJSONItems(items: [[String: AnyObject]]) ->
[MHPointFeatureClusterFeature] {
        var features = [MHPointFeatureClusterFeature]()
        for item in items {
            guard let label = item["itemLabel"] as? [String: AnyObject],
                  let title = label["value"] as? String else { continue
}
            guard let coor = item["coor"] as? [String: AnyObject],
                  let point = coor["value"] as? String else { continue }

            let parsedPoint = point.replacingOccurrences(of: "Point(",
with: "").replacingOccurrences(of: ")", with: "")
            let pointComponents = parsedPoint.components(separatedBy: "
")
            let coordinate = CLLocationCoordinate2D(latitude:
Double(pointComponents[1])!, longitude: Double(pointComponents[0])!)
            let feature = MHPointFeatureClusterFeature()
            feature.coordinate = coordinate
            feature.title = title
            // A feature's attributes can used by runtime styling for
things like text labels.
            feature.attributes = [
                "name": title,
            ]
            features.append(feature)
        }
        return features
    }
}
```