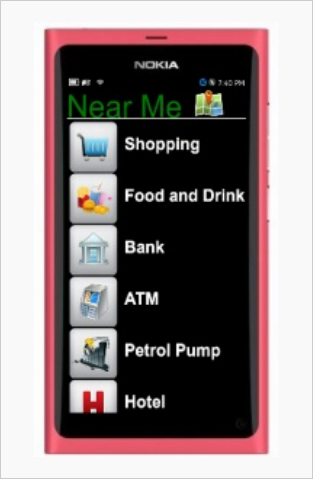
Google Landmarks on a Nokia Map using Qt Quick

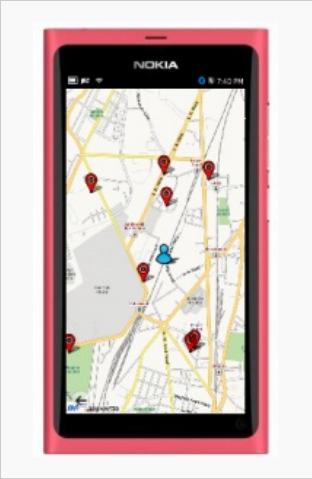
This article demonstrates how to plot landmarks on a Nokia Map using Nokia Map API for Qt.

Introduction

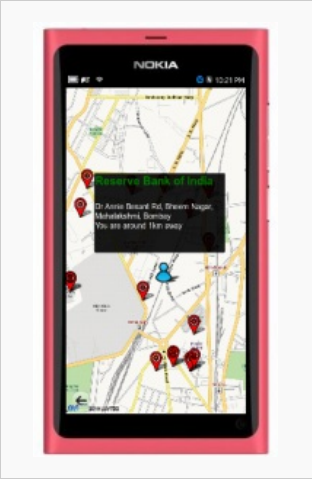
The example combines gets information about local landmarks using the Google Place API and plots them on a Nokia Map using pushpin . We create the pushpins dynamically using Dynamic Object Management in QML. When user clicks on any of the pushpins the application displays a popup with the landmark information and the distance of the landmark from user's location.



Land Mark Category



Pushpin on Landmarks



Land Mark information detail

Implementation

We create two page items in the `main.qml` file. The first page item lists available landmark categories while the second page displays the Nokia Map with the landmarks.

page 1 in main.qml

```
//page 1
    Rectangle {
        id: page1
        width: parent.width
        color: "black"
        height: parent.height
        RssModel {id: feedModelLandmark}
        ListView {
        id: list
        width:  parent.width ;
        height: parent.height-111;
        y:111
        model: feedModelLandmark
        delegate: CategoryListDelegate {}
         }
        Rectangle{
            id:titleRect
            color: "white"
            Text {
                id: tileText
                width:300
                height: 40
                y:32
                text: qsTr("Near Me")
                color: "green"
                font.pixelSize: 80
                }
         }
              Image {
                  id: titleImage
                  source: "maps-icon.png"
                  height: 80
                  width: 80
                  y:30
                  x:340
              }
        Rectangle
        {
            id:underline
            color: "white"
            y:106
            height: 2
            width:parent.width
        }
     states:
            State {
            name: "Item"; when: appWindow.inFirstScreen == true
            PropertyChanges { target: page1; x: 0 }
```


(C) Copyright Nokia 2013. All rights reserved.

```
            PropertyChanges { target: page2; x: -(parent.width * 1.5) }
             PropertyChanges { target: commonTools;       visible:false }
                    }
        transitions: Transition {
            NumberAnimation { properties: "x"; duration: 700; easing.type: Easing.InOutQuad }
        }
     }
```

page 2 in main.qml

```
//page 2
    Item {
        id: page2
        width: parent.width
        height: parent.height
        //display the map
        Rectangle{
                  id: mapRect
                  anchors.fill: parent
                  signal xmlMapReady
                  onXmlMapReady: container.xmlReady = !container.xmlReady
                  RssModelMap {id: feedModelMap}
                  MapDelegate {id: container }
             }
       states:
            State {
            name: "Item"; when:  appWindow.inSecondScreen  == true
            PropertyChanges { target: page2; x: 0 }
            PropertyChanges { target: page1; x: -(parent.width * 1.5) }
            PropertyChanges { target: commonTools;       visible:true }
                    }
        transitions: Transition {
            NumberAnimation { properties: "x"; duration: 700; easing.type: Easing.InOutQuad }
        }
    }
```

The page also has a "back" button, which is visible when the Map displays

```
// menu bar
    ToolBarLayout {
        id: commonTools
        visible: false
        ToolIcon {
            platformIconId: "toolbar-back"
            anchors.bottom: (parent === undefined) ? undefined : parent.bottom
            onClicked: {
                loadFirst()
            }
        }
    }
```

When an user clicks on any of the landmark categories the application calls the `loadLandMark()` function, which uses the Google Places API to get the near by landmark information. To access the Google Places API please get the key from here and replace **<Your Google API Key>** with the key in the below code.

```
function loadLandMark(landmarkname)
    {
        myLatitude =  positionSource.position.coordinate.latitude;
        myLongitude = positionSource.position.coordinate.longitude;
       console.log("latitude:" +myLatitude + "longitude:" +myLongitude+ "landmark:"+landmarkname );
       var url =
           "https://maps.googleapis.com/maps/api/place/search/xml?location="+myLatitude+","+myLongitude+"&radius=1000&name="+landmar
        var doc = new XMLHttpRequest();
        doc.onreadystatechange = function() {
        if (doc.readyState == XMLHttpRequest.HEADERS_RECEIVED) {
          }
        else if (doc.readyState == XMLHttpRequest.DONE) {
             console.log("HttpRequest DONE");
             var status = doc.status;
             if(status!=200) {
                return;
            }
            var data = doc.responseText;
            //console.log("Google Data:\n"+data);
        feedModelMap.xmlData = data;
         }
        }
        doc.open("GET", url);
        doc.send();
    }
```

QML Map Element has a default model called LandmarkModel which provides a model of landmarks from the landmarks store. But in this case we will use XmlListModel instead of LandmarkModel for storing the landmarks data. The Google Places API is a `HTTPS` request so, first we download all the landmarks data through `loadLandMark()` function and then pass the data to the **xml** property of XmlListModel leaving the **source** property empty.

RssModelMap.qml

```
XmlListModel {
    id: feedModelMap
    property string xmlData:"";
     source:""
     xml: xmlData
     query: "/PlaceSearchResponse/result"
     namespaceDeclarations:  ""
     XmlRole { name: "name"; query: "name/string()" }
     XmlRole { name: "vicinity"; query: "vicinity/string()" }
     XmlRole { name: "type"; query: "type/string()" }
     XmlRole { name: "lat"; query: "geometry/location/lat/string()" }
     XmlRole { name: "lng"; query: "geometry/location/lng/string()" }
     XmlRole { name: "reference"; query: "reference/string()" }
     XmlRole { name: "id"; query: "id/string()" }
```


(C) Copyright Nokia 2013. All rights reserved.

```
    onStatusChanged: {
            if (status === XmlListModel.Ready) {
                console.log("xmlModel OK - count =", count);
                mapRect.xmlMapReady();
            }
        }
}
```

When the data loading is finished the XmlListModel generates a signal which then fetches landmark information from the XmlListModel and calls the function `addPushPin()` to plot the coordinates on the Map dynamically.

MapDelegate.qml

```
onXmlReadyChanged: {
                    console.log("******", feedModelMap.count);
                    var locationCount = feedModelMap.count;
                    for(var i=0; i<locationCount; i++ )
                    {
                        var name = feedModelMap.get(i).name;
                        var address = feedModelMap.get(i).vicinity
                        var latitude =feedModelMap.get(i).lat;
                        var longitude =feedModelMap.get(i).lng;
                        console.log("******",name, latitude, longitude);
                        map.addPushPin(latitude, longitude,  name, address);
                    }
                }
```

`addPushPin()` function is our point of attraction, here we create dynamic pushpin for each landmarks using Dynamic Object Management in QML. `addPushPin()` function creates instances of `PushPin.qml` component dynamically. Each instance is a MapImage with properties *latitude*,*longitude*,*name* and *address*. We also put all the pushpin object into an Array, so that we can have a track of all the pushpin plotted on the Map. The method addMapObject() adds the map object on the Map.

```
            function addPushPin( latitude, longitude,name,address ) {
                        var myArray
                         var count = map.markers.length
                         counter++
                var component = Qt.createComponent("PushPin.qml");
                console.log(component.status + " " + Component.Null);
                var object = component.createObject(container,
                                                    { "latid": latitude,  "longit": longitude,"name":name, "address":address}
                                                    );
                if (object.status == Component.Ready) {
                    console.log("Component ready");
                    map.addMapObject(object);
                    object.clicked.connect(pushPinClicked);
                      myArray = new Array()
                            for (var i = 0; i<count; i++){
                                myArray.push(markers[i])
                            }
                            myArray.push(object)
                            markers = myArray
                }
                if (object.status == Component.Error) {
                    console.log("Error: " + component.errorString());
                }
            }
```

PushPin.qml

```
MapImage {
    id:mapMarker
    width: 35
    signal clicked(double lat, double lon,string name, string address)
    source:  "pushpin.png"
    property alias latid: mapMarkerCordinate.latitude
    property alias longit: mapMarkerCordinate.longitude
    property string name:""
    property string address:""
    coordinate: Coordinate
    {
        id: mapMarkerCordinate;
    }
    MapMouseArea {
        onClicked:  mapMarker.clicked(coordinate.latitude, coordinate.longitude,name,address)
    }
}
```

When users click on the pushpin it displays a popup message with landmark's name, address and the approximate distance of the landmark from user location.

MapDelegate.qml

```
function pushPinClicked(aLat, aLng,aName,aAddress)
            {
                myLatitude =  positionSource.position.coordinate.latitude;
                myLongitude = positionSource.position.coordinate.longitude;
                 var dist= distance(myLatitude,myLongitude,aLat,aLng);
                popup.text = aName + "\n";
                popup.text1 =aAddress;
                popup.text2 ="You are around "+dist+" away";
                popup.posX = 250;
                popup.posY = 350;
                popup.show();
            }
```

Every time we select a new landmark from the category list, we need to make sure the pushpins of the previous landmark should not be there in the Map. To do this we need to delete all the previous pushpins from the Map before plotting pushpins for the new landmark. The `deleteAllPushPin()` method does

(C) Copyright Nokia 2013. All rights reserved.

the same, it calls the `destroy()` method to delete each pushpin item dynamically.

```
                function deleteAllPushPin(){
                         var count = map.markers.length
                         for (var i = 0; i<count; i++){
                             map.removeMapObject(map.markers[i])
                             map.markers[i].destroy()
                         }
                         map.markers = []
                         counter = 0
                     }
```

The application also has the Pinch Zoom In/Out feature. The user can go back with the back key and select other landmarks from the category.

Summary

This is a basic Nokia Map API operation. This article also focus on how to use XmlListModel instead of LandmarkModel in Map element, handling HTTPS request from Google Places API and Creating a Map Object Dynamically using Dynamic Object Management in QML.

Source Code

The full source code of the example is available here: File:GoogleLandmarksQt v 1.1.zip

Related Article

- Display Facebook Friend position with Nokia Maps API for Qt

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.