ArcGIS SDK

Localización, rutas y subir archivos Shape y Raster

Localización

Librerías

- #include "LocationDisplay.h"
- #include "MapViewTypes.h"

Archivo Ruta.cpp

```
MapQuickView *Ruta::mapView() const
{
    return m_mapView;
}

void Ruta::localizacion(){
    m_mapView->locationDisplay()->start();
    // centrar la visualización de ubicación alrededor de la ubicación del dispositivo
    m_mapView->locationDisplay()->setAutoPanMode(LocationDisplayAutoPanMode::Recenter);
}


// Establecer vista (creada en QML)
void Ruta::setMapView(MapQuickView *mapView)
{
    if (!mapView || mapView == m_mapView) {
        return;
    }

    m_mapView = mapView;
    m_mapView->setMap(m_map);

    localizacion();

    emit mapViewChanged();
}
```

Nota: el método localización() debe asignarse (como privado) en la clase de la app en el archivo .h

Ruteo

Archivo Ruta.h

```
#ifndef RUTA_H
#define RUTA_H

namespace Esri::ArcGISRuntime {
class Map;
class MapQuickView;

class Graphic;
class GraphicsOverlay;
class PictureMarkerSymbol;
class RouteTask;

} // namespace Esri::ArcGISRuntime

enum RouteBuilderStatus
{
   NotStarted,
   SelectedStart,
   SelectedStartAndEnd,
};

#include <QObject>

#include "RouteParameters.h"
class QAbstractListModel;
Q_MOC_INCLUDE("QAbstractListModel")

Q_MOC_INCLUDE("MapQuickView.h")
```

Archivo Ruta.h

```
Q_MOC_INCLUDE("MapQuickView.h")

class Ruta : public QObject
{
  Q_OBJECT

  Q_PROPERTY(Esri::ArcGISRuntime::MapQuickView *mapView READ mapView WRITE setMapView NOTIFY
        mapViewChanged)
  Q_PROPERTY(QAbstractListModel* directions MEMBER m_directions NOTIFY directionsChanged)
public:
  explicit Ruta(QObject *parent = nullptr);
  ~Ruta() override;

signals:
  void mapViewChanged();
  void directionsChanged();

private:
  Esri::ArcGISRuntime::MapQuickView *mapView() const;
  void setMapView(Esri::ArcGISRuntime::MapQuickView *mapView);
  void localizacion();
  void setupRouteTask();
  void findRoute();
  void resetState();
  Esri::ArcGISRuntime::Map *m_map = nullptr;
  Esri::ArcGISRuntime::MapQuickView *m_mapView = nullptr;
  Esri::ArcGISRuntime::GraphicsOverlay* m_graphicsOverlay = nullptr;
  Esri::ArcGISRuntime::RouteTask* m_routeTask = nullptr;
  Esri::ArcGISRuntime::Graphic* m_startGraphic = nullptr;
  Esri::ArcGISRuntime::Graphic* m_endGraphic = nullptr;
  Esri::ArcGISRuntime::Graphic* m_lineGraphic = nullptr;
  QAbstractListModel* m_directions = nullptr;
  Esri::ArcGISRuntime::RouteParameters m_routeParameters;
  RouteBuilderStatus m_currentState;
};
#endif // RUTA_H
```

Librerías

- #include "Map.h"
- #include "MapQuickView.h"
- #include "MapTypes.h"
- #include "LocationDisplay.h"
- #include "MapViewTypes.h"

- #include "Point.h"
- #include "Viewpoint.h"
- #include "SpatialReference.h"
- #include <QFuture>
- #include "DirectionManeuverListModel.h"
- #include "Graphic.h"
- #include "GraphicListModel.h"
- #include "GraphicsOverlay.h"
- #include "GraphicsOverlayListModel.h"
- #include "Polyline.h"
- #include "RouteTask.h"
- #include "RouteResult.h"
- #include "RouteParameters.h"
- #include "Route.h"
- #include "SimpleLineSymbol.h"
- #include "SimpleMarkerSymbol.h"
- #include "Stop.h"
- #include "Symbol.h"
- #include "SymbolTypes.h"
- #include <QGeoPositionInfoSource>
- #include <QList>
- #include <QUrl>
- #include <QUuid>

Archivo Ruta.cpp

```
using namespace Esri::ArcGISRuntime;

Ruta::Ruta(QObject *parent /* = nullptr */)
    : QObject(parent)
    , m_map(new Map(BasemapStyle::ArcGISStreets, this))
    ,m_currentState(RouteBuilderStatus::NotStarted)
    {
    setupRouteTask();
    }

Ruta::~Ruta() {}

MapQuickView *Ruta::mapView() const
{
    return m_mapView;
}
```

Archivo Ruta.cpp

```
 return m_mapView;
}


void Ruta::localizacion(){
   m_mapView->locationDisplay()->start();
   // centrar la visualización de ubicación alrededor de la ubicación del dispositivo
   m_mapView->locationDisplay()->setAutoPanMode(LocationDisplayAutoPanMode::Recenter);

connect(m_mapView, &MapQuickView::mouseClicked, this, [this](QMouseEvent& mouse)
        {
           const Point mapPoint = m_mapView->screenToLocation(mouse.position().x(), mouse.position().y());
           switch (m_currentState)
           {
           case RouteBuilderStatus::NotStarted:
              resetState();
              m_currentState = RouteBuilderStatus::SelectedStart;
              m_startGraphic->setGeometry(mapPoint);
              break;
           case RouteBuilderStatus::SelectedStart:
              m_currentState = RouteBuilderStatus::SelectedStartAndEnd;
              m_endGraphic->setGeometry(mapPoint);
              findRoute();
              break;

           case RouteBuilderStatus::SelectedStartAndEnd:
              // Ignore touches while routing is in progress
              break;
           }
        });
```

... continua

```
}
```

Archivo Ruta.cpp

... continuación

```
        case RouteBuilderStatus::SelectedStartAndEnd:
          // Ignore touches while routing is in progress
          break;
        }
      });


  m_graphicsOverlay = new GraphicsOverlay(this);
  m_mapView->graphicsOverlays()->append(m_graphicsOverlay);

  SimpleLineSymbol* startOutlineSymbol = new SimpleLineSymbol(SimpleLineSymbolStyle::Solid, QColor("blue"), 2/*width*/, this);
  SimpleMarkerSymbol* startSymbol = new SimpleMarkerSymbol(SimpleMarkerSymbolStyle::Diamond, QColor("orange"), 12/*width*/, this);
  startSymbol->setOutline(startOutlineSymbol);
  m_startGraphic = new Graphic(this);
  m_startGraphic->setSymbol(startSymbol);

  SimpleLineSymbol* endOutlineSymbol = new SimpleLineSymbol(SimpleLineSymbolStyle::Solid, QColor("red"), 2/*width*/, this);
  SimpleMarkerSymbol* endSymbol = new SimpleMarkerSymbol(SimpleMarkerSymbolStyle::Square, QColor("green"), 12/*width*/, this);
  endSymbol->setOutline(endOutlineSymbol);
  m_endGraphic = new Graphic(this);
  m_endGraphic->setSymbol(endSymbol);

  SimpleLineSymbol* lineSymbol = new SimpleLineSymbol(SimpleLineSymbolStyle::Solid, QColor("blue"), 4/*width*/, this);
  m_lineGraphic = new Graphic(this);
  m_lineGraphic->setSymbol(lineSymbol);

  m_graphicsOverlay->graphics()->append(QList<Graphic*> {m_startGraphic, m_endGraphic, m_lineGraphic});

}
```

Archivo Ruta.cpp

```
 m_graphicsOverlay->graphics()->append(QList<Graphic*> {m_startGraphic, m_endGraphic, m_lineGraphic});
//------------------------------
}


void Ruta::setupRouteTask()
{
   // crea la tarea de ruta que apunta a un servicio en línea
   m_routeTask = new RouteTask(QUrl("https://route-api.arcgis.com/arcgis/rest/services/World/Route/NAServer/Route_World"), this);

   // Cree los parámetros predeterminados que cargarán la tarea de ruta implícitamente.
   m_routeTask->createDefaultParametersAsync().then(this,[this](const RouteParameters& routeParameters)
                                    {
                                       // Almacene los parámetros de ruta resultantes.
                                       m_routeParameters = routeParameters;
                                    });
}
```

Archivo Ruta.cpp

```
void Ruta::findRoute()
{
    if (m_routeTask->loadStatus() != LoadStatus::Loaded || m_routeParameters.isEmpty())
        return;

    // Establezca parámetros para devolver direcciones.
    m_routeParameters.setReturnDirections(true);

    // Borrar paradas anteriores de los parámetros.
    m_routeParameters.clearStops();

    // Establezca las paradas según los parámetros.
    const Stop stop1(Point(m_startGraphic->geometry()));
    const Stop stop2(Point(m_endGraphic->geometry()));
    m_routeParameters.setStops(QList<Stop>{ stop1, stop2 });

    // Resuelve la ruta con los parámetros.
    m_routeTask->solveRouteAsync(m_routeParameters).then(this,[this](const RouteResult& routeResult)
                                 {
                                     // Agregue el gráfico de ruta una vez que se complete la resolución.
                                     const Route generatedRoute = routeResult.routes().at(0);
                                     m_lineGraphic->setGeometry(generatedRoute.routeGeometry());
                                     m_currentState = RouteBuilderStatus::NotStarted;

                                     // Establecer el modelo de lista de maniobras de dirección.
                                     m_directions = generatedRoute.directionManeuvers(this);
                                     emit directionsChanged();
                                 });
}
```

Archivo Ruta.cpp

```
void Ruta::resetState()
{
    m_startGraphic->setGeometry(Point());
    m_endGraphic->setGeometry(Point());
    m_lineGraphic->setGeometry(Point());
    m_directions = nullptr;
    m_currentState = RouteBuilderStatus::NotStarted;
}
```

Archivo Ruta.cpp

```
void Ruta::setMapView(MapQuickView *mapView)
{
    if (!mapView || mapView == m_mapView) {
        return;
    }

    m_mapView = mapView;
    m_mapView->setMap(m_map);

    localizacion();

    emit mapViewChanged();
}
```

Archivo RutaForm.qml

```
import QtQuick
import QtQuick.Controls
import Esri.Ruta
//------------------------------
import QtQuick.Shapes
//------------------------------
Item {//Apertura del Item


    // Cree MapQuickView aquí y cree su mapa, etc. en código C++
    MapView {
        id: view
        anchors.fill: parent
        // set focus to enable keyboard navigation
        focus: true
    }

    // Declarar la instancia de C++ que crea el mapa, etc. y proporcionar la vista
    Ruta {
        id: model
        mapView: view
    }
```

... continua

Archivo RutaForm.qml

... continuación

```
// Crear ventana para mostrar las direcciones de ruta.
    Rectangle {
        id: directionWindow
        anchors {
            right: parent.right
            top: parent.top
            margins: 5
        }

        radius: 5
        visible: model.directions
        width: Qt.platform.os === "ios" || Qt.platform.os === "android" ? 250 : 350
        height: parent.height / 2
        color: "#FBFBFB"
        clip: true

        ListView {
            id: directionsView
            anchors {
                fill: parent
                margins: 5
            }
            header: Component {
                Text {
                    height: 40
                    text: "Directions:"
                    font.pixelSize: 22
                }
            }

            // Establezca el modelo en DirectionManeuverListModel devuelto por la ruta.
            model: model.directions
            delegate: directionDelegate
        }
    }
```

... continua

Archivo RutaForm.qml

... continuación

```
Component {
        id: directionDelegate
        Rectangle {
            id: rect
            width: parent.width
            height: textDirections.height
            color: directionWindow.color

            // separador de dirección
            Shape {
                height: 2
                ShapePath {
                    strokeWidth: 1
                    strokeColor: "darkgrey"
                    strokeStyle: ShapePath.SolidLine
                    startX: 20; startY: 0
                    PathLine { x: parent.width - 20 ; y: 0 }
                }
            }
```

... continua

Archivo RutaForm.qml

... continuación

```
Text {
            id: textDirections
            text: qsTr("%1 (%2 miles)".arg(directionText).arg((length * 0.00062137).toFixed(2)))
            wrapMode: Text.WordWrap
            anchors {
                leftMargin: 5
                left: parent.left
                right: parent.right
            }
        }
    }
}

}//Cierre del Item
```

Archivos shape y raster

Librerías

- #include "Map.h"
- #include "MapQuickView.h"
- #include "MapTypes.h"
- #include "LocationDisplay.h"
- #include "MapViewTypes.h“

- #include "Point.h"
- #include "Viewpoint.h"
- #include "SpatialReference.h"
- #include <QFuture>
- #include "DirectionManeuverListModel.h"
- #include "Graphic.h"
- #include "GraphicListModel.h"
- #include "GraphicsOverlay.h"
- #include "GraphicsOverlayListModel.h"
- #include "Polyline.h"
- #include "RouteTask.h"
- #include "RouteResult.h"
- #include "RouteParameters.h"
- #include "Route.h"
- #include "SimpleLineSymbol.h"
- #include "SimpleMarkerSymbol.h"
- #include "Stop.h"
- #include "Symbol.h"
- #include "SymbolTypes.h"
- #include <QGeoPositionInfoSource>
- #include <QList>
- #include <QUrl>
- #include <QUuid>

- #include "ShapefileFeatureTable.h"
- #include "FeatureLayer.h"
- #include "LayerListModel.h"

- //#include "Raster.h"
- //#include "RasterLayer.h "

- #include "SimpleRenderer.h"
- #include "SimpleMarkerSymbol.h"
- #include "SymbolTypes.h"

Archivo Ruta.cpp

```
void Ruta::CargarCapas()
{

   //const Point center(-75, 5, SpatialReference::wgs84());
   //const Viewpoint viewpoint(center, 100000.0);
   //m_mapView->setViewpointAsync(viewpoint);

   QString shapefilePath = "D:/DOCS/ASIGNATURAS IMPARTIDAS/ArcGIS SDK/data/Col/gadm41_COL_1.shp";
   ShapefileFeatureTable* shapefileFeatureTable  = new ShapefileFeatureTable(shapefilePath, this);
   FeatureLayer* featureLayer = new FeatureLayer(shapefileFeatureTable, this);
   m_map->operationalLayers()->append(featureLayer);

   SimpleRenderer* renderer = new SimpleRenderer(new SimpleMarkerSymbol(SimpleMarkerSymbolStyle::Circle, QColor("red"), 10.0), this);
   featureLayer->setRenderer(renderer);


}
```

```
void Ruta::localizacion(){
```

Nota: el método CargarCapas() debe asignarse (como privado) en la clase de la app en el archivo .h

Archivo Ruta.cpp

```
void Ruta::setMapView(MapQuickView *mapView)
{
    if (!mapView || mapView == m_mapView) {
        return;
    }

    m_mapView = mapView;
    m_mapView->setMap(m_map);

    localizacion();
    CargarCapas();

    emit mapViewChanged();
}
```