MapKit3D Qt
Getting Start Guide

Version 1.03

Copyright

Address

TeleCommunication Systems, Inc.
6A Liberty
Aliso Viejo, Ca 92656 USA

(949) 453-1646
www.telecomsys.com

Trademarks

Contents

1 Overview

1.1 Introduction

This *Getting Start Guide* provides basic information to create applications using the MapKit3D library from the LocationToolkit (LTK) on PC and Linux based embedded systems. After reading this guide, you should be able to create a sample application that uses the LTK libraries to display and control a map.

1.2 Assumptions

This document assumes that you are familiar with Qt and C++ development. Before getting start, you should install the Qt libraries on your computer. If you are working on the Windows platform, you will also need to install MinGW as the compiler environment. This SDK makes use of OpenGL to render the map, make sure that the OpenGL libraries are correctly installed and Qt's OpenGL support is enabled. We recommend that you install and use QtCreator to facilitate faster development.

For development environment setup and required Qt libraries, please refer to:

LTK Qt Linux Dev Environment Setup.pdf

For runtime environment setup and required Qt libraries, please refer to:

LTK Qt Linux Run Environment Setup.pdf

We strongly recommend you to read through these documents before you get started.

2 Getting Started

This section explains how to run the Qt sample project provided with LTK. You will also learn how to integrate LTK MapKit3D into your Qt project.

2.1 Run the MapKit3D Sample Project

2.1.1 Unzip package

Unzip the release package ltk-<version>.zip, the unzipped folder structure looks like:

<release_package_folder>

+----assets

+----bin

| +----mapkit3d_sample

| +----navkit_sample

+----docs

+----library

| +----include

| +----lib

| +----locationtoolkit.pro

+----samples

| +----mapkit3d_sample

| +----navkit_sample

2.1.2 Compile the MapKit3D Sample Code

1. Start QtCreator and click the “Open Project” button on the welcome page
2. Choose the project file found in release_package_folder/samples/mapkit3d_sample/project.
3. In the edit page of QtCreator, right click on mapkit3d_sample and choose “build” in the menu.

2.1.3 Run the sample app

Once the compile finishes, the created output file “mapkit3d_sample.exe” can be found at:

samples/mapkit3d_sample/project/ debug(or release).

Copy the output file into release_folder/bin/mapkit3d_sample.

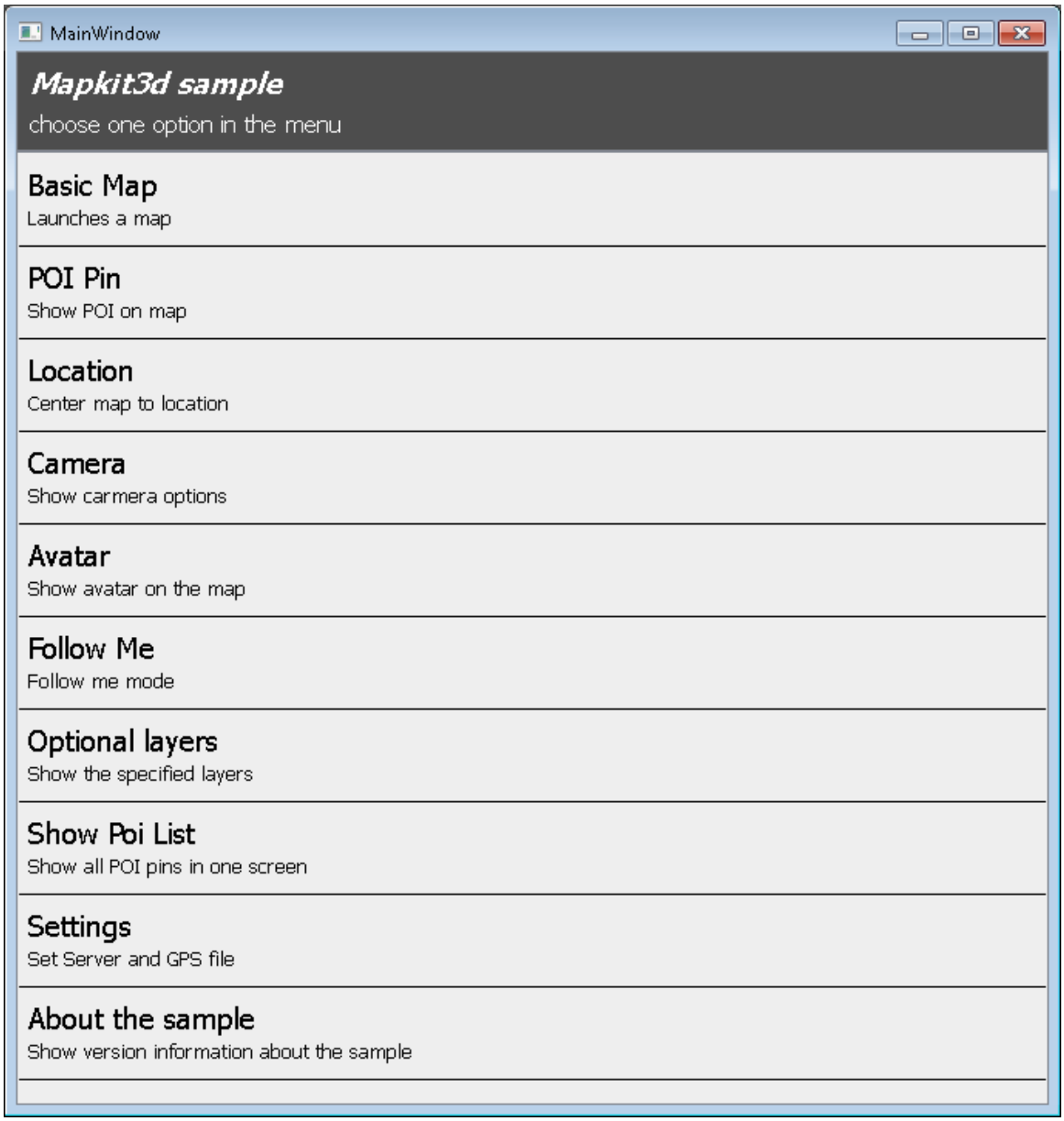
<@TBD this doesn’t look right for Linux, or complete>

On Windows, just double click the exe file. On Linux, type this command:

sh startup.sh

The following window will display on successful execution:

Error! Reference source not found.



2.1.4 Gesture Controls in Map

Using mouse and keyboard, you can simulate gesture controls in map

Gesture	32Ctrl	Alt	Shift	Mouse Left Button	Map
Pan	N/A	N/A	N/A	Slide up	Move up
				Slide down	Move down
				Slide left	Move left
				Slide right	Move right
Scale	N/A	N/A	Press	Slide up	Zoom out
				Slide down	Zoom in
Rotate	Press	Press	N/A	Slide CW	Rotate CW
				Slide CCW	Rotate CCW
Tilt	Press	N/A	N/A	Slide up	Tilt angle smaller
				Slide down	Tilt angle bigger
Double Tap	N/A	N/A	N/A	Double click	Zoom in
Two finger tap	Press	N/A	N/A	Double click	Zoom out
wheel	N/A	N/A	N/A	Roll up	Zoom in
				Roll down	Zoom out

2.2 Build your Own Application Using MapKit3D

2.2.1 Create Your Project

For simplicity, start with the QtCreator wizard. For example, create a Qt application with the mainwindow wizard, then modify the pro file to include the LTK SDK into your project:

a. You must declare a variant named LTK_ROOT which points to the library path in the SDK, and then include locationtoolkit.pro into your own project file. The code will be similar to the following:

```
LTK_ROOT = some path…/ltk_1.0.0.84/library
include ($$LTK_ROOT/locationtoolkit.pro)
```

b. Remove the auto generated code, it will look similar to the following:

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

The locationtoolkit.pro already contains these lines.

After these steps, your project file should contain two parts, one is the declaration of LTK_ROOT and includes the pro file in the SDK, the other part references your own .h, .cpp and .ui files, as seen here:

Figure2 project file sample

sample.pro

```
#declare LTK_ROOT and include template
LTK_ROOT = ../../ltk_1.0.0.84/library
include ($$LTK_ROOT/locationtoolkit.pro)

# reference to your own files
INCLUDEPATH += ../include

SOURCES += \
    ../src/main.cpp \
    ../src/mainwindow.cpp \
    ../src/toolbarcommon.cpp \
    ../src/basicmap/basicmapwidget.cpp \
    ../src/basicmap/toolbarbasicmap.cpp \

HEADERS  += \
    ../include/mainwindow.h \
    ../include/inifile.h \
    ../include/toolbarcommon.h \
    ../include/basicmap/basicmapwidget.h \
    ../include/basicmap/toolbarbasicmap.h \


FORMS    += \
    ../Forms/mainwindow.ui \
    ../Forms/positioninputdialog.ui \

```

Figure2 project file sample

2.2.2 Create the LTKContext Object

You need to create an LTK Context object first when using this SDK. There are seven parameters needed, refer to the API documentation for details. The code will look similar to the following:

```
ltkContext = new locationtoolkit::LTKContext(credential,
                                             "qt",
                                             "en-US",
                                             0000000000ULL,
                                             domain,
                                             hostname,
                                             dpi );
```

2.2.3 Create a Map Widget Object

<@TBD need better English, I don't know the intent here>

Create the widget will new operator and put it into some layout in the main window. Note that when you create the map widget object, do not set any parent widget in the constructor. Code may like:

```
mMapWidget = new locationtoolkit::MapWidget();
QVBoxLayout* pLayout = new QVBoxLayout;
pLayout->addWidget( mMapWidget );
```

2.2.4 Initialize the Map Widget

Use the mapOption object to initialize the map widget. The code will look similar to the following:

```
locationtoolkit::MapOptions mapOption;
mapOption.mEnableFullScreenAntiAliasing = true;
mapOption.mEnableAnisotropicFiltering = true;
mapOption.mResourceFolder = "./resource/";
mapOption.mWorkFolder = "./";
mMapWidget->Initialize( mapOption,  *ltkContext); // the LtkContext object you created in 2.2.2
```

The map member options are described in the following table:

Member	Means
mEnableFullScreenAntiAliasing	Enable/disable anti-aliasing
mEnableAnisotropicFiltering	Enable/disable anisotropic filtering
mResourceFolder	Resource filepath. Should be same as the SDK release folder/bin/mapkit3d_sample
mWorkFolder	The path where you want to store temporary and downloaded files

Now that you have created and initialized your map widget, you can now compile and run the application. Please note that before you start your own application, you must have a resource folder and point to it using mResourceFolder. The structure of the folder should be the same as that found in release_path/bin/mapkit3d_sample.

On successful execution, you will see a window similar to the one below:

Figure 3 map view

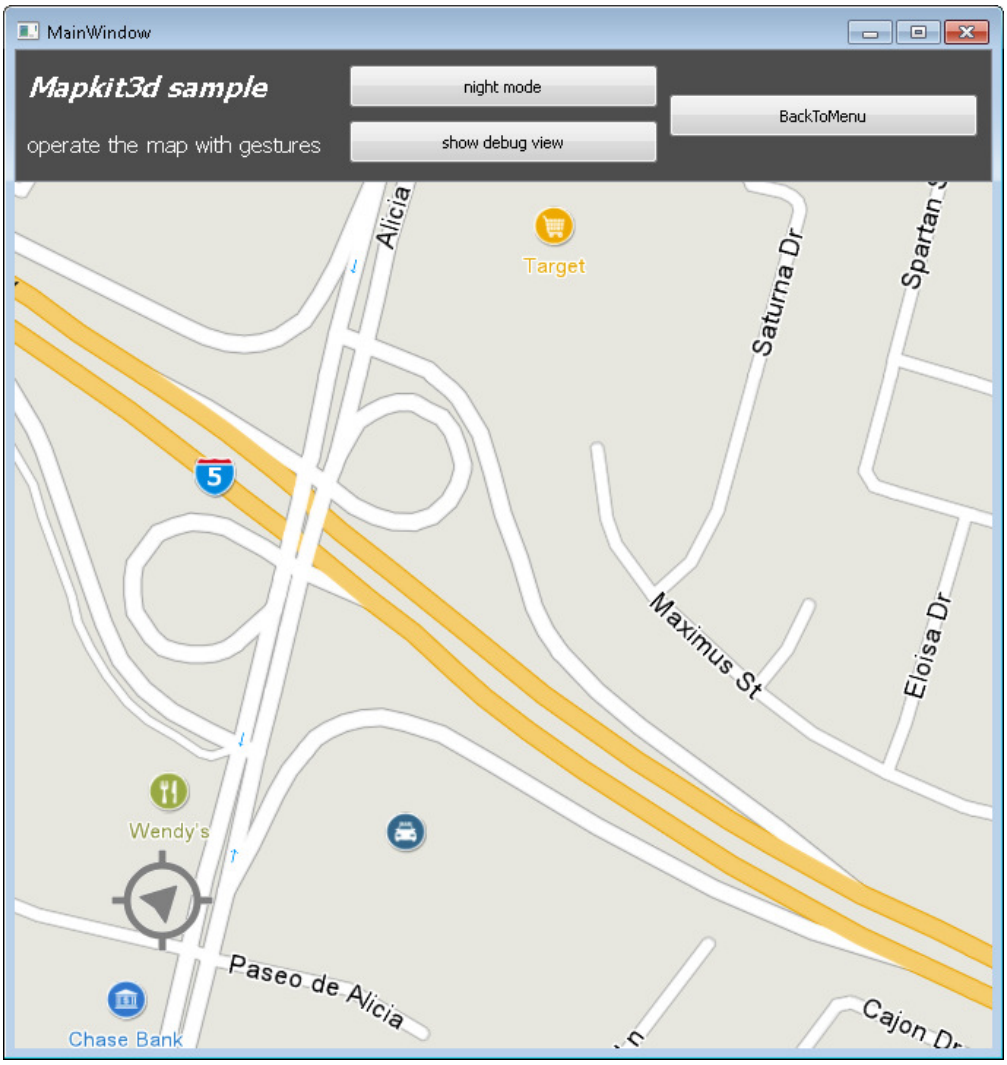


Figure 3 map view

3 MapKit3D Programming Guidelines for Qt Development

3.1.1 Change Location

Use the MoveTo function to move the camera to a location using latitude and longitude:

```
double movetolat = 33.76985;
double movetolon =;
locationtoolkit::Coordinates coordin( movetolat, movetolon );
locationtoolkit::CameraParameters param( coordin );
mMapWidget->GetCameraPosition( param );
param.SetPosition( coordin );
    mMapWidget->MoveTo( param ); -118.19302
```

3.1.2 Change Zoom Level

Use the MoveTo function to set the camera zoom level:

```
locationtoolkit::Coordinates coordin(33.76985, 33.76985);
    locationtoolkit::CameraParameters param( coordin );
    mMapWidget->GetCameraPosition( param );
    param.SetZoomLevel( 12.0 );
    mMapWidget->MoveTo( param );
```

3.1.3 Zoom to a Bounding Box

Use the ZoomToBoundingBox to zoom to a specific bounding box

```
mMapWidget->ZoomToBoundingBox( minLatitude, minLongitude, maxLatitude, maxLongitude );
```

3.1.4 Use Animation

Use AnimateTo to animate smooth map movement to a place or a zoom level. Create a locationtoolkit::AnimationParameters object to indicate the animation properties first, then call the AnimatoTo function.

```
locationtoolkit::AnimationParameters mAniParam( locationtoolkit::AnimationParameters::AC_Deceleration, 1000 );
locationtoolkit::CameraParameters param( coordin );
// set the parameters you want to param object
mMapWidget->AnimateTo( param, mAniParam );
```

3.1.5 Place an Avatar on the map

Use SetLocation to place an avatar at a specific location.

```
locationtoolkit::Location defaultLocation = {0};
defaultLocation.latitude = 33.604;
defaultLocation.longitude = -117.689;
defaultLocation.heading = 60.0;
defaultLocation.valid = 507;
mMapWidget->GetAvatar().SetMode( locationtoolkit::Avatar::AM_CAR );
mMapWidget->GetAvatar().SetLocation( defaultLocation );
```

3.1.6 Switch from day/night mode

Set map to day or night mode by calling SetNightMode

```
mMapWidget->SetNightMode( locationtoolkit::MapWidget::NM_DAY );  or
mMapWidget->SetNightMode( locationtoolkit::MapWidget::NM_NIGHT );
```

3.1.7 Set compass position

Set the compass on the map to a specific screen position, ex. (300, 40)

```
mMapWidget->SetCompassPosition( 300, 40 );
```

3.1.8 Put pins on the map

Place Points of Interest (POI) pins on the map with user defined pin images

```
mMapWidget->RemoveAllPins();  // remove other pins, optional
locationtoolkit::PinImageInfo selectedImage;
selectedImage.SetPixmap(a QPixmap object containing the image when the pin is selected);
locationtoolkit::PinImageInfo unSelectedImage;
unSelectedImage.SetPixmap(a QPixmap object containing the image when the pin is not selected);
locationtoolkit::RadiusParameters radiusPara(50, 0x6721D826);
locationtoolkit::PinParameters pinpara(
                coordinate,  // the coordinate you want to put pin
    selectedImage,       // PinImageInfo object of pin's selected image
    unSelectedImage,     // PinImageInfo object of pin's unselected image
    radiusPara,          // the radius of the pin's background shadow
    title,               // title text
    subtitle,            // sutitle text
    bubble,                          // pointer to bubble object. If NULL, uses the
                                     // default bubble
    visible);            // true, of course
mMapWidget->AddPin(pinpara);
```

Note: A "bubble" is the small widget showing various information when you click on the pin. Users can specify their own bubble by writing a class derived from the Bubble interface.

3.1.9 Set Follow-me mode

Use SetGPSMode function to let the camera follow an avatars position automatically

```
mMapWidget->SetGpsMode(locationtoolkit::MapWidget::GM_FOLLOW_ME);
```

The available parameters are:

GM_FOLLOW_ME: camera will follow avatar with avatar's head points up

GM_FOLLOW_ME_ANY_HEADING: camera will follow the avatar from a fixed angle. Allows the avatar's head to point in any direction

GM_FOLLOW_STAND_BY: camera does not follow the avatar

3.1.10 Draw a Polyline

<@TBD I need help deciphering this opening sentence>

Drawing a polyline in map should provide the following parameters before call AddPolyline

- A list of coordinates indicates all the points
- A list of segment attributes indicates the color of the point
- <@TBD is “Tow” a typo?>
- Tow CapParameter objects indicate how to draw the start and end point of each line segment
- Width of the polyline
- Other colors of the polyline, such as highlighted color or outline color

```
QList<locationtoolkit::Coordinates> mPolyPoints;
  QList<locationtoolkit::SegmentAttribute>* mSegAttr;
     ……

  // prepare poly line points and segments
  for_each( point, PointsOfPolyline )
  {
       mPolyPoints.append( point );
       locationtoolkit::ColorSegment segattr(mPolyPoints.count()-1, QColor(Qt::green));
       mSegAttr->append(*segattr);
  }

  // prepare start and end cap parameters
  int polylineWidth = 10;
  CapParameter mStartCap;
  mStartCap.type = locationtoolkit::CPT_Round;
  mStartCap.radius = polylineWidth /2;
  mStartCap.width = 0;
  mStartCap.length = 0;
  CapParameter mEndCap;
  mEndCap.type = locationtoolkit::CPT_Round;
  mEndCap.radius = polylineWidth /2;
  mEndCap.width = 0;
  mEndCap.length = 0;

  // create polyline parameter
  polylineParameters para;
  para.SetPoints(mPolyPoints).SetSegmentAttributes(*mSegAttr).SetUnhighlightColor(QColor(Qt::green)).SetWidth(polylineWid
  th).SetZOrder(16).SetVisible(true).SetStartCap(mStartCap).SetEndCap(mEndCap).setOutlineColor(QColor(Qt::gray)).setOutline
  Width(5);

  // draw the polyline
  mMapWidget->AddPolyline(para);
```

3.1.11 Display optional layers in map

Turn on or off additional layers (satellite, traffic and doppler) by setting the layer.enabled attribute

```
const QList<locationtoolkit::MapWidget::LayerNameAndEnabled> layers = mMapWidget->GetLayerNameAndStates();
for( int i = 0; i < layers.size(); i++ )
{
      // current layer name is layers[i].name and
    // layers[i].enabled indicates whether it is shown now
      layers[i].enabled = true;
}
```

To display traffic layer, call ShowTrafficLayer:

```
mMapWidget->ShowTrafficLayer( QBool(true/false) );
```

To display other optional layers, call ShowOptionalLayer

```
mMapWidget->ShowOptionalLayer( layerName, show )
```