[TOC]

开放平台Demo使用说明

一、资源文件配置

1.将/opt/目录设置为可读写权限 chmod u+rw opt/   
2.将bnav中的资源文件放到/opt/BaiduMapAuto/bnav/目录下（此目录可配置，请修改BaiduMapAutoQmlInterface.cpp中CFG\_DATA\_CONFIG变量）离线数据更新请按照bnav/ReadMe.txt文档更新 

二、Linux环境安装依赖库

1.安装编译工具集 安装命令：

sudo apt-get install build-essential

安装完成后验证gcc／g++版本 PS: if meet ” E: Unable to locate package essential” ,please update apt-get sudo apt-get update

2.安装OpenGL工具集 安装opengl 3命令：

sudo apt-get install freeglut3-dev

安装opengl es命令：

sudo apt-get install libgles2-mesa-dev

3.安装freetype 更新索引列表

apt-get update  
sudo apt-get install libfreetype6-dev

三、Demo工程配置

建立so软链接：进入 automap-linux-qml/libs/ 目录 ，执行 sh ln\_sh.sh

功能模块使用说明

* 开发环境：Ubuntu14.04 + Qt 5.7.0，使用qml开发UI, 工程名字automap-linux-qml.pro
* ubuntu 环境搭建 http://agroup.baidu.com/automap\_linux/md/article/152991
* 引擎使用流程简介：（请结合BaiduMapAutoQmlInterface.cpp） 按功能模块划分为mapView、search、routeGenerator、routeGuide；  
  每个模块都要创建新实例，调用相应的API接口，API详见doc;

mapView模块

mapView实例创建一定放在在SurfaceView创建好之后进行

**1、初始化底图**

BDMapViewer\* m\_pMap;  
 m\_pMap = new BDMapViewer(BDMapType::FRONT, DEV\_WINDOW\_WIDTH, DEV\_WINDOW\_HEIGHT);  
Parameters  
[in] type map type such as front, rear left and so on  
[in] width Sub-Surface's width value  
[in] height Sub-Surface's height value  
//设置地图显示level  
m\_pMap->setMapLevel(14);  
//设置地图中心点偏移：比如导航过程中设置车标偏下显示或者显示路口放大图的时候，设置导航路线偏右显示  
m\_pMap->setMapOffset(0，-100);  
//设置日夜景模式  
m\_pMap->setMapTheme(BDMapTheme::NIGHT\_MODE1);

**2、在绘制线程以40ms左右周期调用** mapView->draw()

m\_pMap.draw();

**3、监听消息** 添加监听

m\_pMap->setEventListener(this);

底图等级变化

void MyBDMapViewerListener::onMapPositionChanged(const BDMapViewer& object, const common::BDGeoCoord& coord) {  
 ...  
}  
Parameters  
[in] pos The geo-position of the center of BDMapViewer instance

**4、多边形绘制**

示例代码

//多边形定点定义，坐标为国测局经纬度坐标，并乘以100000，顺时针方向绘制定点  
std::vector<common::BDGeoCoord> geoArr;  
  
 common::BDGeoCoord tmpCoord1;  
 tmpCoord1.setLongitude(11631326);  
 tmpCoord1.setLatitude(4004173);  
 geoArr.push\_back(tmpCoord1);  
  
 common::BDGeoCoord tmpCoord2;  
 tmpCoord2.setLongitude(11633548);  
 tmpCoord2.setLatitude(4003021);  
 geoArr.push\_back(tmpCoord2);  
  
 common::BDGeoCoord tmpCoord3;  
 tmpCoord3.setLongitude(11662187);  
 tmpCoord3.setLatitude(4006176);  
 geoArr.push\_back(tmpCoord3);  
  
 common::BDGeoCoord tmpCoord4;  
 tmpCoord4.setLongitude(11639759);  
 tmpCoord4.setLatitude(3990877);  
 geoArr.push\_back(tmpCoord4);  
  
 common::BDGeoCoord tmpCoord5;  
 tmpCoord5.setLongitude(11629523);  
 tmpCoord5.setLatitude(3998697);  
 geoArr.push\_back(tmpCoord5);  
//样式定义，rgba，值在0~1之间  
m\_pMap->setMapRegionStyle(0.0f, 0.6f, 0.1f, 0.3f);  
//设置数据并开始绘制  
m\_pMap->setMapRegionData(geoArr);  
//清除多边形绘制  
m\_pMap->clearMapRegionData();

底图POI点击（POI详情数据需要进一步修改 ）

**5、自定义底图标注**

示例代码——添加标注点

std::vector<HDynamicMapPoint> array;  
HDynamicMapPoint point0, point1, point2;  
//大厦  
point0.name = "百度大厦";  
point0.point.setLongitude(11630218);  
point0.point.setLatitude(4005007);  
point0.tag = "TYPE\_1";  
array.push\_back(point0);  
//奎科  
point1.name = "奎科科技大厦";  
point1.point.setLongitude(11630691);  
point1.point.setLatitude(4004174);  
point1.tag = "TYPE\_1";  
array.push\_back(point1);  
//科技园  
point2.name = "百度科技园";  
point2.point.setLongitude(11627512);  
point2.point.setLatitude(4004396);  
point2.tag = "TYPE\_2";  
array.push\_back(point2);  
  
m\_pMap->setDynamicMapPoints(array);

示例代码——设置Marker Icon

QImage\* img = new QImage(16, 16, QImage::Format\_RGBA8888);  
int iTWidth = img->width();  
int iTHeight = img->height();  
int iChannel = 4;  
img->fill(QColor(255, 0, 0, 200));  
m\_pMap->setDynamicMapImages("TYPE\_1", (HChar \*)img->bits(), iTWidth, iTHeight, iChannel);  
delete img;

注：引擎会对图片buffer进行深拷贝，因此上层内存需自己管理

search模块

检索区分区域检索和周边检索，通过设置不同的search filter来区别 检索模块支持sug检索

1、初始化Search模块

BDPOISearch\* m\_pSearch;  
m\_pSearch = new BDPOISearch();

2、设置检索filter（检索相关参数属性，sug检索无需设置filter） a、初始化Filter

BDPOISearchFilter filter;

b、filter设置区域检索或者周边检索，两种检索为互斥的，即filter.setAddress(address)和filter.setBoundary(pos, 1000)同时只使用一个。 **区域检索：**

BDAddress address;  
address.setRegionCode(131);//设置区域id为北京  
filter.setAddress(address);

**周边检索：** 要求半径大于0，单位米，代码示例：

BDGeoCoord pos;  
 pos.setLatitude(3672808);  
 pos.setLongitude(11920123);  
 filter.setBoundary(pos, 1000);

c、翻页功能

filter.setPageOption(BDUInt32 curPage, BDUInt32 pageCount)； // curPage:当前页序号， pageCount：每页的检索结果数量  
 BDPOISearch::isLastPage(); // 判断是否为最后一页

3、设置检索的关键字：filter设置setKeyword

filter.setKeyword("天安门");

4、search中设置filter

m\_pSearch->setSearchFilter(filter);

5、发起检索/sug检索

m\_pSearch->search();  
m\_pSearch->searchSug("肯");//检索sug无需设置filter

6、 设置监听获取检索结果回调 重载IBDPOISearchListener，并实现该类中的虚函数方法。代码示例：

class BaiduMapAutoInterface ：public IBDPOISearchListener  
{  
 m\_pSearch->setEventListener(this);  
 //获取检索的结果  
 void onSearchResultUpdate(BDPOISearch\* pSearch, const BDInt32& status, const BDInt32& count)  
 {//逻辑代码  
 }  
 //获取sug（输入联想）的结果  
 void onSearchSugResultUpdate(BDPOISearch\* pSearch)  
 {//逻辑代码  
 }  
}

7、 跨城市检索结果说明 跨城市检索结果会返回目的地所在的城市名字，拿到返回结果的RegionCode，重新发起区域检索。 这种情景的标识为：BDPOIInfo::getUid()为空。 例如：检索“上海 KFC” 结果返回，如果info.getAddress().getRegion()[0]为“”，说明返回的是城市列表，列表包括上海和上海的regioncode，再使用regioncode重新发起区域检索。

8、POI详情检索结果 检索示例 http://dev.mapauto.baidu.com/detail?qt=infauto&uid=784961807213499085

9、检索结果地图气泡展示

1）根据以上步骤发起检索，等待检索结果通知 2）获取检索结果列表

searchApi->getResultItemCount(count);  
int offset = 0;  
std::vector<BDPOIInfo> resultList;  
searchApi->getResultItemList(offset, count, resultList);

3）将检索结果的坐标存入vector

std::vector<BDGeoCoord> coords;  
for(int i = 0; i < resultList.size(); ++i)  
{  
 coords.push\_back(resultList[i].getPosition());  
}

4）调用底图接口添加气泡，检索结果为红色气泡，红色气泡标有poi的index

m\_pMap->addBkgPoiMarker(coords);

10、屏幕上点击poi

1）捕获屏幕的点击事件，获取屏幕坐标，调用mapview接口getSelectItemInfo得到poiInfo

QPoint curPoint = event->pos();  
 BDPOIInfo info;  
 m\_pMap->getSelectItemInfo(curPoint.rx(), curPoint.ry(), info);

2)实现两个消息回调，在回调函数中调用选择poi的接口addPoiMarker(common::BDGeoCoord& coord); 在调用getSelectItemInfo时会调用相应的回调函数，选中后，底图上该点会弹出蓝色气泡

* onMapPoiClicked为底图上的poi点
* onMapBkgPoiMarkerClicked为检索结果的红色气泡的poi点

注：若点击底图空白区域则获取不到poi信息，因此没有气泡显示

void MapViewerListener::onMapPoiClicked(const BDMapViewer &object, const BDPOIInfo &info)  
 {  
 map->addPoiMarker(info.getPosition());  
 }  
 void MapViewerListener::onMapBkgPoiMarkerClicked(const BDMapViewer &object, const BDUInt32 &index)  
 {  
 map->addPoiMarker(resultList[index].getPosition());  
 }

11、隐藏气泡

m\_pMap->hideBkgPoiMarker(); //隐藏红色气泡  
m\_pMap->hidePoiMarker(); //隐藏蓝色气泡

routeGenerator模块

1、初始化

BDRouteGenerator\* m\_pRouteGenerator;  
m\_pRouteGenerator = new BDRouteGenerator();

2、设置起点，终点，代码示例

// 1st set startPosition  
 BNACoordinate cur\_pos;  
 AMGeoTools::locate(cur\_pos.iLongitude, cur\_pos.iLatitude);  
 BDGeoCoord start\_geo\_coord;  
 start\_geo\_coord.setLatitude(cur\_pos.iLatitude \* 100000);  
 start\_geo\_coord.setLongitude(cur\_pos.iLongitude \* 100000);  
 BDRoutePosition start\_pos;  
 start\_pos.setPosition(start\_geo\_coord);  
 m\_pRouteGenerator->setStartPosition(start\_pos);  
  
// 2nd set destPosition  
BDGeoCoord dest\_geo\_coord;  
dest\_geo\_coord.setLatitude(latitude \* 100000);  
dest\_geo\_coord.setLongitude(longitude \* 100000);  
BDRoutePosition dest\_pos;  
dest\_pos.setPosition(dest\_geo\_coord);  
m\_pRouteGenerator->setDestination(dest\_pos);

3、设置检索参数，通过routeOption来设置

BDRouteOption routeOption;  
std::vector<std::string> optionList;  
//获取可设置的参数list   
routeOption.getOptionList(optionList);  
//设置为高速优先  
routeOption.setOption("BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_HIGHWAY",1);  
//设置为在线检索模块  
routeOption.enableOnlineMode(true);  
//设置车牌号，本地化限行  
routeOption.setOption("BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_CARNUM",1);  
BDCarInfo carinfo("P261U6", "津", BDCarTypeEnum::BD\_CAR\_TYPE\_14L);  
m\_pRouteGenerator->setLocalCarInfo(carinfo);  
//将设置option给Generator  
m\_pRouteGenerator->addRouteOption(routeOption);

算路偏好使用说明： -BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_RECOMMEND 推荐路线 - BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_HIGHWAY 高速优先 - BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_NO\_TOLL 少收费 - BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_AVOID\_TRAFFICJAM 躲避拥堵 - BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_CARNUM 车牌限行 eg: 以下设置是高速优先和躲避拥堵同时生效 routeOption.setOption(“BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_HIGHWAY”, 1); // 高速优先， 第二个参数为1时生效，为0时不生效 routeOption.setOption(“BNA\_ROUTEPLAN\_PREFERENCE\_TYPE\_AVOID\_TRAFFICJAM”, 1)； 注意：*高速优先和少收费是互斥关系；车牌限行和其它项同时使用；推荐路线只能和车牌限行同时使用；躲避拥堵除不能和推荐一起使用外，可以和其它项同时使用*

4、设置途经点

BDRoutePosition wpt\_pos;  
 BDGeoCoord geo\_coord;  
 geo\_coord.setLatitude( lat );  
 geo\_coord.setLongitude( lon );  
 wpt\_pos.setPosition( geo\_coord);  
 m\_pRouteGenerator->addWaypoint(wpt\_pos);

下次算路前要清除途经点：

m\_pRouteGenerator->clearWaypoints();

5、发起算路

m\_pRouteGenerator->generate();

6、算路结果监听并绘制在底图上 重载IBDRouteGeneratorListener，并实现该类中的虚函数方法onGenerateStatusUpdated（）。代码示例：

class BaiduMapAutoInterface ：public IBDRouteGeneratorListener  
{  
m\_pRouteGenerator->setEventListener(this);  
// 获取路径规划结果  
void BaiduMapAutoInterface::onGenerateStatusUpdated(const BDRouteGenerator& generator, const BDRouteGeneratingStatus& status, const BDByte& numOfRoutes)  
{  
 //获取返回的路线数量  
 int route\_cnt = numOfRoutes > 3 ? 3: numOfRoutes;  
 BDGeoArea geo\_area;  
 BDRouteInfo route\_info;  
 //获取道路路线数组给mapview显示  
 std::vector<baidu::mapauto::navi::route::BDRouteInfo> routeInfos;  
 for (int i = 0; i < route\_cnt; i++)  
 {  
 int index = i + 1;  
 m\_pRouteGenerator->getRouteInfo(i, route\_info);  
 routeInfos.push\_back(route\_info);  
 emit setRoutePlanResultItemData(index, route\_info.getTotalTime(),  
 route\_info.getTotalDistance(), route\_info.getTrafficLight());  
 }  
  
 // 在mapView添加路线  
 m\_pMap->addRoutes(routeInfos);  
  
 // 调整底图显示比例  
 geo\_area = route\_info.getArea();  
 baidu::mapauto::common::BDRectangle rect = {0, 0, 800, 600};  
 m\_pMap->fitGeoAreaToScreen(geo\_area, rect);  
 }  
}

routeGuide模块

1、初始化

BDRouteGuide\* m\_pRouteGuide  
m\_pRouteGuide = BDRouteGuide::getInstance();  
m\_pRouteGuide->setBasicEventListener(this);  
m\_pRouteGuide->setViewEventListener(this);

2、开启导航或开启模拟导航 （1）开启导航：

BDResult result = m\_pRouteGuide->startRouteGuidance();

（2）开启模拟：

BDResult result = m\_pRouteGuide->startSimulation();

3、模拟导航设置速度：

float speed = 10; //>0,speed+; <0, speed-  
m\_pRouteGuide->setSimulationSpeed(10);

4、TBT消息处理

void onTurnInfoUpdated(const std::vector<BDTurnInfo>& turnInfoList)  
{  
 BDTurnInfo turn\_info = turnInfoList[0];  
  
// Next road name  
QString next\_road\_name = QString::fromStdString(turn\_info.getNextRoadName());  
  
// Next road distance  
int neext\_road\_distance = turn\_info.getRemainDistance();  
  
// Remain time  
int route\_index = 0;  
BDUInt32 remain\_time;  
m\_pRouteGuide->getRemainTime(route\_index, remain\_time);  
  
// Remain distance  
BDUInt32 remain\_distance;  
m\_pRouteGuide->getRemainDistance(route\_index,remain\_distance);  
}

5、路口放大图 （1）消息通知后，更新过程

void onILSImageViewUpdated(const BDILSImageViewInfo& hInfo) {  
 //get buffer  
 const char\* pBGMap = hInfo.getBackGoundMap().c\_str();  
 const char\* pArrowMap = hInfo.getArrowMap().c\_str();  
 unsigned int bgLength = 0;  
 unsigned int arrowLength = 0;  
 unsigned char\* pbgByteBuf = NULL;  
 unsigned char\* parrowByteBuf = NULL;  
 m\_pRouteGuide->getRasterExpandMapImage(pBGMap, 1, &pbgByteBuf, bgLength);  
 m\_pRouteGuide->getRasterExpandMapImage(pArrowMap, 0, &parrowByteBuf, arrowLength);  
 //load file  
 ...  
 }  
（

2）关闭路口放大图：收到下面的消息并收起路口放大图

void onHideILSImageViewUpdated(bool& status) {  
}

6、智能比例尺

// Close autoLevel  
BDResult result1 = m\_pRouteGuide->setMapMemoryScale(BNAScreenCenter, level);  
BDResult result2 = m\_pRouteGuide->setMapAutoLevelStatus(BNAScreenCenter, false);

7、路况数据：绘制光柱图

std::vector<BDRoadCondition> arrRoadCondition;  
float progess;  
BDResult res1 = m\_pRouteGuide->getRoadCondition(arrRoadCondition);  
BDResult res2 = m\_pRouteGuide->getCarProgress(progess);

8、停车场处理逻辑 （1）确认收到消息

void onDestParkAvailableUpdated(bool& status) {}

（2）发起停车场信息检索 （3）检索结果返回后，显示在地图上，并拼接出停车场推荐的文本，进行语音播报 （4）当用户点击“停车场”并确定“停在这里”时，获得当前点选停车场的位置 （5）重新设定起点为当前点位置：通过GetMapMatchInfo拿到 （6）设置终点为停车场位置 （7）发起算路 （8）调用switchRoute切换到算路结果中的第一条数据

9、全览态功能 （1）进入全览态：

m\_pRouteGuide->enterViewAll(SIMPLE\_LAND);  
/\*\*< SIMPLE\_LAND参考BDRouteGuideViewAllType \*/

（2）退出全览态：

m\_pRouteGuide->exitViewAll();

10、巡航模式 （1）未开启导航模式下，地理位置连续变化数达到一定阈值并且速度达到一定阈值，会抛出可以开启消息

onRouteCruiseAvailableUpdated();

（2）根据需要可以调用接口，开启巡航

startRouteCruise();

（3）在一定条件（如用户操作底图）下，可以调用接口关闭巡航

stopRouteCruise();

11、主辅路切换功能： （1）主辅路的切换及消隐都是通过消息控制

onChangeRouteUpdated(BDRouteGuideChangeRouteType resChangeRoute);

其中，BDRouteGuideChangeRouteType各定义如下：

CHANGETO\_MAINROUTE, //切换到主路  
CHANGETO\_SALVEROUTE, //切换到辅路  
CHANGETO\_PARALLELROUTE, //切换到平行路线  
HIDECHANGEROUTE, //隐藏切换信息

（2）调用接口完成道路切换

m\_pRouteGuide->onlineChangeRoute();

12、获取当前路名(底图非导航模式使用，导航模式推荐onRoadNameUpdated())：

m\_pRouteGuide->getCurRoadNameByPos(longitude, latitude);

其中longitude、latitude格式为gcj02 \* 100000

13、在导航中路线刷新

m\_pRouteGuide->refreshRoute();

刷新完成后在onRefreshRouteUpdated回调方法中返回刷新结果，结果类型说明：

typedef enum \_BDRefreshRouteStatusType {  
REFRESH\_ROUTE\_INVALID = -1, /\*\*< invalid\*/  
REFRESH\_ROUTE\_SUCCESS, /\*\*< refresh success\*/  
REFRESH\_ROUTE\_FAILED, /\*\*< refresh failed\*/  
REFRESH\_ROUTE\_NEWROUTE, /\*\*< update new Route\*/  
REFRESH\_ROUTE\_NO\_NEWROUTE, /\*\*< no Route\*/  
REFRESH\_ROUTE\_NET\_TIMEOUT, /\*\*< timeout\*/  
REFRESH\_ROUTE\_OTHER\_ROUTE /\*\*< other route\*/  
}BDRefreshRouteStatusType;

Location模块

Location模块功能包括两部分： 1. 支持LocationManager的状态操作，如开启、暂停、恢复、停止，以及更换LocationDriver进行模拟导航等。 2. 支持同步和异步获取Location相关信息。

1、初始化和销毁Location模块：

// 初始化  
BDLocation\* m\_pLocation;  
m\_pLocation = BDLocation::getInstance();  
m\_pLocation->setEventListener(listener);  
// 销毁  
m\_pLocation->unsetEventListener();

2、操作LocationManager的运行状态：

m\_pLocation->start(); // 开启Location线程  
m\_pLocation->stop(); // 结束Location线程  
m\_pLocation->pause(); // 暂停Location线程  
m\_pLocation->resume(); // 恢复Location线程  
// 切换成NMEA和NaviTrack模拟器进行模拟导航。(interval是采样时间，单位是us；speed单位是m/s)  
m\_pLocation->setNmeaSimulator(trackFile， interval, speed));  
m\_pLocation->setNaviTrackSimulator(trackFile, interval, speed));  
// 切换成算路结果进行模拟导航，暂未实现  
m\_pLocation->setRouteSimulator(); // 暂未实现  
// 切换会正常导航模式（目前支持passive driver，如果不同平台，需要单独配置，如ccos driver）  
m\_pLocation->unsetSimulator(); // 目前只支持passive driver

3、同步获取Location相关信息：

// 同步获取Location结构  
BDLocationInfo LocationInfo；  
BDResult ret = m\_pLocation->getLocationInfo(LocationInfo);  
if (ret == BDResult::ERROR) {  
 // bdlog\_e 记录出错信息  
}

4、异步获取Location相关信息：

// 异步获取Location结构  
class LocationApi : public IBDLocationListener {  
 BDLocationInfo m\_clLocationInfo；  
   
 void onLocationInfoUpdated(const BDLocationInfo& LocationInfo) {  
 // 周期触发  
 // bdlog\_v 记录LocationInfo输出数据  
 doSomeThingWithLocationInfo(LocationInfo);  
 }  
}  
m\_pLocationApi = new LocationApi();  
m\_pLocation->setEventListener(m\_pLocationApi);  
// onLocationInfoUpdated triggered periodly

MapMatching模块

MapMatching模块实现了巡航态、导航态下MapMatching数据输出的功能，以及惯导数据输出的功能。 目前并未实现惯导功能，只提供了惯导数据输出接口。

1、初始化和销毁MapMatching模块：

// 初始化  
BDMapMatching\* m\_pMapMatching;  
m\_pMapMatching = BDMapMatching::getInstance();  
m\_pMapMatching->setEventListener(listener);  
// 销毁  
m\_pMapMatching->unsetEventListener();

2、同步获取MapMatching相关信息：

// 同步获取MapMatching结构  
BDMapMatchingInfo MapMatchingInfo；  
BDResult ret = m\_pMapMatching->getMapMatchingInfo(MapMatchingInfo);  
if (ret == BDResult::ERROR) {  
 // bdlog\_e 记录出错信息  
}

4、异步获取MapMatching相关信息：

// 异步获取MapMatching结构  
class MapMatchingApi : public IBDMapMatchingListener {  
 BDMapMatchingInfo m\_MapMatchingInfo；  
   
 void onMapMatchingInfoUpdated(  
 const BDMapMatchingInfo& MapMatchingInfo) {  
 // 周期触发  
 // bdlog\_v 记录MapMatchingInfo输出数据  
 doSomeThingWithMapMatchingInfo(MapMatchingInfo);  
 }  
}  
m\_pMapMatchingApi = new MapMatchingApi();  
m\_pMapMatching->setEventListener(m\_pMapMatchingApi);  
// onMapMatchingInfoUpdated triggered periodly