密级：机密

文档编号： 第 版

分册名称： 第 册/共 册

匹配路

接口设计参考

北京高德泰信科技有限公司

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 总页数 |  | 正文 |  | 附录 | 无 | 生效日期 |  |
| 编制 | 孙冬艳 | | | 审批 |  | | |

**目 录**

[1 文档概述 1](#_Toc318125082)

[1.1 介绍 1](#_Toc318125083)

[1.2 文档版本 1](#_Toc318125084)

[1.3 参考文档 1](#_Toc318125085)

[2 接口设计需求 1](#_Toc318125086)

[3 接口设计 1](#_Toc318125087)

[3.1 数据定义 1](#_Toc318125088)

[3.1.1 命名空间 1](#_Toc318125089)

[3.1.2 接口函数返回值定义 1](#_Toc318125090)

[3.1.3 回调函数定义 2](#_Toc318125091)

[3.1.4 数据结构定义 2](#_Toc318125092)

[3.2 接口函数设计 3](#_Toc318125093)

[3.2.1 开始gps校正处理接口 3](#_Toc318125094)

[3.2.2 结束一次校正处理 3](#_Toc318125095)

[3.2.3 用户输入gps点 3](#_Toc318125096)

[3.2.4 开始轨迹到路段匹配 3](#_Toc318125097)

[3.2.5 结束轨迹到路段匹配 4](#_Toc318125098)

[3.2.6 提交gps轨迹 4](#_Toc318125099)

[4 接口使用 4](#_Toc318125100)

[4.1 调用前的准备 4](#_Toc318125101)

[4.1.1 数据准备 4](#_Toc318125102)

[4.1.2 静态库准备 6](#_Toc318125103)

[4.1.3 调用过程 6](#_Toc318125104)

[4.1.4 注意事项 7](#_Toc318125105)

[5 示例程序 7](#_Toc318125106)

# 文档概述

## 介绍

本文档是针对匹配路的接口设计及使用说明。

## 文档版本

|  |  |  |
| --- | --- | --- |
| 版本号 | 修改时间 | 说明 |
| 0.1 | 2012-2-24 | 将匹配路封装成一组API |
|  |  |  |

## 参考文档

|  |  |  |
| --- | --- | --- |
| 序号 | 文档名 | 在VSS上的位置 |
| 1 | 地图接口定义 |  |

# 接口设计需求

见《地图接口定义》

# 接口设计

见《地图接口定义》

## 数据定义

### 命名空间

接口的命名空间为“MathchRoadApi”。

### 接口函数返回值定义

返回值为void类型。

### 回调函数定义

//函数: void (\*)(GpsPoint\_t\* pts, size\_t size)

//功能: 地图轨迹匹配完成回调返回给用户的接口

//参数: pts –轨迹点地址; size –轨迹点数量

//返回: true –最后一个轨迹点，意味用户已结束轨迹转换

//注：pts的内存由程序分配并回收，用户深拷贝。

typedef void (\_\_cdecl \*CallBack\_PointFixed)(GpsPoint\_t\* pts, size\_t size);

//函数: void (\*)(RoadPart\_t\* parts, size\_t size)

//功能: 地图接口返回匹配的路段信息到用户的接口注：Parts数据由地图接口库完成内存分配和释放，用户层不能缓存parts数据，必须进行深拷贝

//参数: parts –路段信息; size –路段数量

//返回: None

//注：parts的内存由程序分配并回收，用户深拷贝。

typedef void (\_\_cdecl \*CallBack\_NodeSplitted)(RoadPart\_t\* parts, size\_t size);

调用约定采用标准C/C++的。

### 数据结构定义

1、GpsPoint\_t

//描述一个gps轨迹点信息

typedef struct

{

float lon; //经度

float lat; //纬度

int timetick; //Unix timestamp 1970 ~ 秒为单位

float speed; //速度也许并不能获取

float angle; //角度也许并不能获取

void \*Delta; //用户层私有数据作为上下文传递 (Fileoffset Gps时间对应影像文件偏移量)

}GpsPoint\_t,\*pGpsPoint\_t;

**注意：**GpsPoint\_t中：

lon及lat，为为便宜的经纬度坐标（度）；

Timetick的单位为秒。

Speed无速度的时候为0值。

Angle无角度的时候为-1值。如果有角度，是以正北为基准顺时针。

Delta为用户开辟的内存的地址。

2、RoadNode\_t

//描述一个路段节点

typedef struct

{

int Mesh\_id; //图幅号

int Node\_id; //节点编号

}RoadNode\_t,\*pRoadNode\_t;

**注意**：RoadNode\_t中：

Mesh\_id为整型的图幅编号。

3、RoadPart\_t

//描述一个路段

typedef struct

{

RoadNode\_t first\_node; //路段的两端节点中的开始节点 注：gps轨迹跨越路段具有方向性

RoadNode\_t second\_node; //路段的两端节点中的结束节点 注：gps轨迹跨越路段具有方向性

GpsPoint\_t First\_point; //出现在（first，second）路段上第一个gps坐标点

GpsPoint\_t Last\_point; //出现在（first，second）路段上最后一个gps坐标点

}RoadPart\_t,\*pRoadPart\_t;

## 接口函数设计

### 开始gps校正处理接口

函数: void gpsfix\_begin(CallBack\_PointFixed user);

功能: 开始gps校正处理，并实现初始化地图数据。

参数: user –完成gps修复之后传递给用户的接收接口

返回: void

### 结束一次校正处理

函数:　void gpsfix\_end();

功能: 结束一次校正处理,匹配接口接收到用户调用gpsfix\_end之后，将内部残留的gps点即刻全部通过CallBack\_PointFixed返回给用户，并且再追加调用一次CallBack\_PointFixed函数，并设置pts为NULL,或者size 为，以便用户层可以获知已经调用gpsfix\_end()。

参数: 无

返回: void

### 用户输入gps点

函数: void gpsfix\_data(GpsPoint\_t\* pt,size\_t ptsize)

功能: 用户输入gps坐标点。此函数被用户连续调用，gps数据流式的被要求提交进处理模块。

参数: pt –坐标点地址; ptsize –坐标点数量

返回: void

### 开始轨迹到路段匹配

函数: void nodesplit\_begin(CallBack\_NodeSplitted user);

功能: 开始轨迹到路段匹配

参数: user –用户接收接口

返回: void

### 结束轨迹到路段匹配

函数: void nodesplit\_end();

功能: 结束gps轨迹到路段的匹配，地图SDK内部完成最后一次的处理工作，通过CallBack\_NodeSplitted接口返回所有的匹配路段。并且再追加调用一次CallBack\_NodeSplitted函数，并设置parts为NULL,或者size 为，以便用户层可以获知已经调用nodesplit\_end()。

参数: none

返回: none

### 提交gps轨迹

函数: void nodesplit\_data(GpsPoint\_t\* pts, size\_t size);

功能: 用户提交dvr采集的gps轨迹数据到地图接口。Gps轨迹是连续提交给地图接口，地图接口内部实现缓冲，完成处理之后通过callback\_nodesplitted反射回用户层。

参数: pts –gps轨迹点数组

Size - 轨迹点数量

返回: none

# 接口使用

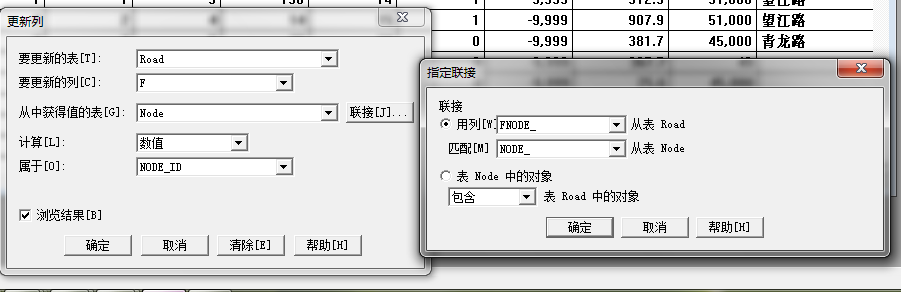
## 调用前的准备

### 数据准备

用户调用前必须确定工作环境下有RoadMap数据文件夹，该文件夹下有拓扑数据文件以及Road\_Node.txt文本文件。其中，拓扑数据文件是由我部门提供的。而Road\_Node.txt是由用户生成的。生成Road\_Node.txt的方法如下：

用户需安装mapinfo软件，并且需要road.tab及node.tab两份数据。并把这两份数据做好备份，因为以下操作会修改两个tab表。在mapinfo中打开两份数据：

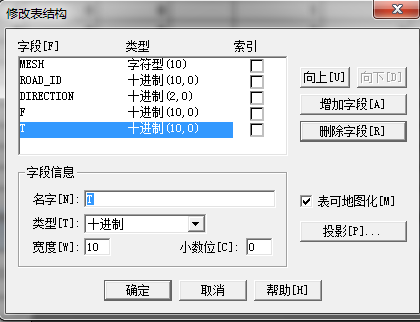
1. 表->维护->表结构：增加两列分别为F，T。都为十进制 10 0类型。
2. 表->更新列：界面输入如下所示：



1. 点击确定，后，再点击：表->更新列，操作如下图所示：



1. 然后点击：表->维护表结构。选择road表，确定，对弹出的对话框做如下修改：



其中，字段的顺序不能颠倒。然后点击确定、确定。

1. 然后点击表->转出->选择road->转出。保存名称为“Road\_Node.txt”，保存类型为txt。确定。分界符勾选其他，分界符为半角的英文逗号，文件字符集默认即可，不要勾选以第一行作为列标题。点击确定即可。

至此，所需要的文件转换完成。

### 静态库准备

MatchRoadApi.h、MatchRoadApi.lib、XLong.lib。

### 调用过程

1、校正处理的调用过程

void cb\_gpsdata(GpsPoint\_t\* pts, size\_t size){

这里接收修正的gps数据

If ( pts == NULL ){

已经到达Gpsfix\_end

}

}

Gpsfix\_begin(cb\_gpsdata);

GpsPoint\_t \* pt;

while ( pt = readgps() ){

Gpsfix\_data(pt,1);

}

Gpsfix\_end();

开始时，用户初始化地图模块，调用gpsfix\_begin()函数，

设置好数据返回的入口(CallBack\_PointFixed),连续读入dvr的gps数据，调用gpsfix\_data()将gps轨迹提交给处理模块； 处理模块内部实现缓冲、匹配等处理之后通过CallBack\_PointFixed接口将修正的数据返回给用户层；用户结束输入调用gpsfix\_end()通知处理模块，处理模块完成最后的匹配之后将残存的所有gps轨迹提交给用户(callback\_pointfixed)，并清除自己内部的缓冲，重新初始化自己的状态。

2、匹配接口的调用过程

void cb\_roadparts (RoadPart\_t\* pts, size\_t size){

这里接收路段数据

If ( pts == NULL ){

已经到达nodesplit\_end()

}

}

nodesplit\_begin(cb\_gpsdata);

GpsPoint\_t \* pt;

While( pt = readgps() ){

nodesplit\_data(pt,1);

}

nodesplit\_end();

程序开始调用nodesplit\_begin,设置路段匹配结果接收；

连续读取dvr轨迹数据并通过nodesplit\_data接口传递给地图SDK，地图sdk实现一定的缓存处理之后计算出轨迹跨越的路段通过 callback\_nodesplitted返回给调用层；

程序全部完成轨迹读取之后调用nodesplit\_end通知地图sdk结束道路轨迹匹配工作，地图sdk即刻处理内部所有的gps轨迹数据，通过callback\_nodesplitted返回给用户层, 并清除自己内部的缓冲，重新初始化自己的状态.

### 注意事项

1. 在使用TMCAPI时要注意命名空间，否则编译过程会有问题；
2. 注意回调函数的调用约定。

# 示例程序

以下是一个控制台程序演示API的使用。（详见sample程序）

// sample.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include "./include/MatchRoadApi.h"

#pragma comment ( lib , "./lib/matchroadapi.lib" )

#pragma comment ( lib , "./lib/XLong.lib" )

void cb\_gpsdata(MathchRoadApi::GpsPoint\_t\* pts, size\_t size)

{

for(int i=0;i<size;i++)

{

printf("x=%.6f y=%.6f\n",pts[i].lon,pts[i].lat);

}

}

void cb\_roadparts(MathchRoadApi::RoadPart\_t\* parts, size\_t size)

{

for(int i=0;i<size;i++)

{

printf("firN:mesh=%d node=%d secN:mesh=%d node=%d \n",parts[i].first\_node.Mesh\_id,parts[i].first\_node.Node\_id,parts[i].second\_node.Mesh\_id,parts[i].second\_node.Node\_id );

}

}

void dealGroup1()

{

MathchRoadApi::gpsfix\_begin(cb\_gpsdata);

MathchRoadApi::GpsPoint\_t \* pt=NULL;

pt=new MathchRoadApi::GpsPoint\_t();

float x=0,y=0;

x = 121.561852; y = 31.207855;

pt->angle = -1;

pt->Delta = NULL;

pt->lon = x;

pt->lat = y;

pt->speed = 0;

pt->timetick = 1234567;

MathchRoadApi::gpsfix\_data(pt,1);

delete pt;

MathchRoadApi::GpsPoint\_t \* pt1=NULL;

pt1=new MathchRoadApi::GpsPoint\_t();

x = 121.561859; y = 31.207855;

pt1->angle = -1;

pt1->Delta = NULL;

pt1->lon = x;

pt1->lat = y;

pt1->speed = -1;

pt1->timetick = 1234568;

MathchRoadApi::gpsfix\_data(pt1,1);

delete pt1;

//}

MathchRoadApi::gpsfix\_end();

}

void dealGroup2()

{

MathchRoadApi::nodesplit\_begin(cb\_roadparts);

MathchRoadApi::GpsPoint\_t \* pt=NULL;

pt=new MathchRoadApi::GpsPoint\_t();

float x,y;

x = 121.561852; y = 31.207855;

pt->angle = -1;

pt->Delta = NULL;

pt->lon = x;

pt->lat = y;

pt->speed = 0;

pt->timetick = 1328140800;

MathchRoadApi::nodesplit\_data(pt,1);

delete pt;

MathchRoadApi::GpsPoint\_t \* pt1=NULL;

pt1=new MathchRoadApi::GpsPoint\_t();

x = 121.561859; y = 31.207855;

pt1->angle = -1;

pt1->Delta = NULL;

pt1->lon = x;

pt1->lat = y;

pt1->speed = 0;

pt1->timetick = 1328140850;

MathchRoadApi::nodesplit\_data(pt1,1);

delete pt1;

MathchRoadApi::nodesplit\_end();

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

printf("测试轨迹点修正,开始...\n");

dealGroup1();

printf("测试轨迹点修正,结束\n");

printf("测试匹配路,开始...\n");

dealGroup2();

printf("测试匹配路,结束\n");

return 0;

}