

Fenix 数据编写指南 – Eyderoe

2024/8 ver.1

1. 概要

Fenix 采用 SQLite3 数据库作为数据源，后缀为 db3，Fenix v1 和 v2 在数据库数据部分没有任何差异，数据具体格式为非标准 ARINC424。

一般来说，需要对：Airports 机场，AirwayLegs 航路航段，Airways 航路名，ILSes，NavaidLookup 导航台补充信息，Nav aids 导航台基本信息，Runways 跑道，TerminalLegs 程序基本信息，TerminalLegsEx 程序补充信息，Terminals 程序头，WaypointLookup 航点补充信息，Waypoints 航点基本信息进行编码。

稍微需要一点数据库知识，不多，以下默认对数据库具有一定认知。

Fenix 数据库容忍度很大，有些是必需，有些是最好，有些则是无所谓，后面涉及时会提到。

2. 编写顺序

所有元素（机场，航点，导航台等）都**依靠 ID 进行索引**，而不是采用标准 424 根据类型进行索引，在这部分 Fenix 可以更加灵活的处理信息。注意，ID 为主键只要求唯一而不要求递增。

既然依靠 ID 进行索引，整个数据库需要按一定顺序填入，否则肯定会出现找不到东西的情况，当然在数据库层面上也通过外键进行了约束。

后面章节顺序即为建议顺序。

ID	Ident	Collocated	Name	Latitude	Longitude
1	AARON	0	AARON	-79.3845	-1
2	ANGIE	0	ANGIE	-89.75466667	-1
3	BAKUS	0	BAKUS	-81.43566667	-1
4	BAROW	0	BAROW	-80.21816667	-1
5	BILL	0	BILL	-79.998	-1
6	BLAIR	0	BLAIR	-80.23366667	-1
7	BOHIC	0	BOHIC	-81.86516667	-1
8	BRIAN	0	BRIAN	-81.89016667	-1
9	COREY	0	COREY	-80.18883333	-1
10	DANNY	0	DANNY	-79.3955	-1
11	DEANO	0	DEANO	-79.68283333	-1
12	DEBIE	0	DEBIE	-89.95766667	-1
13	ELIOT	0	ELIOT	-80.24683333	-1
14	ESSPO	0	ESSPO	-79.63366667	-1
15	FABIO	0	FABIO	-79.56166667	-1

图 1 表 Waypoints 的一部分

3. 导航台、航点

导航台和航点是最基础的元素，完成后，依靠传统程序和 MCDU 中的 DIR 功能即可在大多数机场完成导航。

由于 NAIP 和 AIP 有一定重复部分，对于 NAIP 中每一个元素需要先数据库中查询是否存在。建议条件：Ident 相同，Lat 和 Long 对应差值小于 0.2（仅在导航台和非终端区导航台上 NAIP 和 AIP 具有差距）即可认为同一。

首先对导航台 Nav aids 进行编写，大多数参数可以显然得到，不能显然的可以对比已知进行解释，故只对一部分进行解释。

- a. Freq 频率：首先解释 VOR，以江北 CKG 116.1 为例，将 116.1*10,000=1,161,000 作为十六进制读取，其十进制值 18,223,104 即为 Freq 值。同样，对于 NDB 台，以统景场 OS 241 为例，(241*10000)hex=37814272。
- b. SlaveVar 被动磁偏角补偿：地磁有关的问题很麻烦，到处都在吵，这里填 0 就行，但应确保 MagneticVariation 磁差中的值准确。

接着则是 Nav aidsLookup，这里只说明 NavKeyCode 为 1。

因为航点需要与导航台关联，故需要在导航台完成后编写。
航点表中的都很显然。

只说明 Collocated 当航点为导航台时为 1，并应该在 NavaidID 中填写导航台在 Nav aids 表中对应 ID。

有例子如下：

- a. 假设有 NAIP 导航台 GCK，编写导航台表后有 ID=20，且需要继续在航点表中编写航点 GCK，其中的 NavaidID 为 20。
- b. 假设有 NAIP 航点 ABPIN，仅需要在航点表中编写航点 ABPIN。

4. 航路

首先需要将航路名称不在表 Airways 中的添加进去，然后就能愉快的编写航路 AirwayLegs 了。Fenix 的航路采用链式规则编写。对于正常航路有 P 点，涉外航路中有 P 点的复杂情况，对逻辑要求较高，故在附录部分提供了一种简单的算法以实现目的。

链式规则即：A <-> B -> C <-> D 会根据方向被拆分为三条，每一条以 IsStart=1 起始 IsEnd=1 结束。

- a. A -> B -> C -> D
- b. B -> A
- c. D -> C

从航路这里就能看出 Fenix 索引机制，根据 Waypoint1ID 和 Waypoint2ID 在 Waypoint 表中 ID 列寻找，求得对应航点，而不用关心该点是否为导航台。

5. 机场、跑道、ILS

也没有什么好说的其实，只提一下 AirportLookup 中的 extID：ZUCK 为例 extID = ZU + ZUCK = ZUZUCK

对于 ILS，其准确性高度相关于跑道头坐标。**注：此处未验证**

6. 程序

重点内容。

首先每个程序都需要在 Terminal 表中进行注册，Proc 类型：SID-2，STAR-1，APP-3。**对于跑道，建议将多跑道程序拆分为单个跑道进行写入。**比如：ZUCK-STAR-QJG1J 建议拆分写为三条记录 QJG1J-02L，QJG1J-02R，QJG1J-03。

对于程序每条本身，应详细参照 **ARINC424** 和其他单跑道机场已知程序如：ZPLJ 等。

ARINC424-18（导航数据库第 18 版）可以在 **Github 和 Github 或 Gihub 以及一些其他 Github** 上找到。下面给出字段对应内容在 424 中位置。

每一种航段类型需要填的空不一样。一般来说，在数据库编码里列出来的那几项就够了。

- a. Type 程序类型 5.7
- b. Transition 过渡标识 5.11
- c. TrackCode 航段类型 5.21 & P193.ATTACHMENT5
- d. 航点显然，而导航台是指的航点本身是否为导航台，**而不是指推荐导航台 5.23（这里看不懂最好别看）**
- e. TurnDir 转向 5.20
- f. Course 航向 5.26
- g. Distance 距离 5.27 假如是 RF 航段，这里是弧段长，而不是弧半径。
- h. Alt 高度限制 5.29 5.30 假如是 MAPt 点此处强制填“MAP”
- i. WptDescCode 不管 Fenix 好像不读这个“E”

在进近中 IF 前的航段为 RF 时，会导致 RF 弧丢失，需将此 IF 的中心点设置为该 RF 弧中心点。

7. 数据库

数据库部分建议采用 Python 或其他语言编写，附录中给出 Python 示例代码。

由于是数据库操作，频繁查询会导致用时大幅度提高，故同样给出建议的 SQL 新建索引语句以供使用，在 v1 的测试中通过新建索引将效率提高了 90%，而 v2 与 v1 数据库差异也仅在于 v2 内置了索引，当然索引并不改变数据库数据，所以 v1 和 v2 数据库互通。

附录

航路拼接示例代码

```
1. # 使用条件：不会出现分叉
2. master = ['A', 'C', 'F']
3. slave = ['X', 'Y', 'A', 'B', 'C']
4. firstLoc = secondLoc = -1
5. for i in master:
6.     if i in slave:
7.         if firstLoc == -1:
8.             firstLoc = slave.index(i)
9.         else:
10.            secondLoc = slave.index(i)
11.            break
12. if firstLoc > secondLoc:
13.     slave = slave[::-1]
14.
15. insertLoc = 0
16. for i in slave:
17.     if i in master:
18.         insertLoc = master.index(i) + 1
19.     else:
20.         master.insert(insertLoc, i)
21.         insertLoc += 1
22. print(master)
```

数据库操作示例代码

```
1. class Database:
2.     def __init__(self, db_path: str):
3.         """
4.         :param db_path: 数据库路径
5.         """
6.         self.__conn = sqlite3.connect(db_path)
7.         self.__cursor = self.__conn.cursor()
8.
9.     def __del__(self):
10.         self.__conn.commit()
11.         self.__conn.close()
12.
13.     def __get_columns_name(self, table_name: str) -> List[str]:
14.         """
15.         获取列名
16.         :param table_name: 表名
17.         :return: 列名(列表)
18.         """
19.         query = "select name from pragma_table_info('{}')".format(table_name)
20.         self.__cursor.execute(query)
```

```

21.         result = self.__cursor.fetchall()
22.         result = [i[0] for i in result]
23.         return result
24.
25.     def commit(self):
26.         """
27.         立即提交更改
28.         """
29.         self.__conn.commit()
30.
31.     def get_columns(self, table_name: str, aims: List[str]) -> List[tuple]:
32.         """
33.         获取列值
34.         :param table_name: 表名
35.         :param aims: 列名(列表)
36.         :return: 查询结果
37.         """
38.         query = "select {} from {}".format(','.join(aims), table_name)
39.         self.__cursor.execute(query)
40.         result = self.__cursor.fetchall()
41.         return result
42.
43.     def get_columns_sp(self, table_name: str, aims: List[str], conditions: dict) -> List[tuple]:
44.         """
45.         以特定条件获取列值
46.         :param table_name: 表名
47.         :param aims: 目标列
48.         :param conditions: 列名:列值
49.         :return: 查询结果
50.         """
51.         aims = ','.join(aims)
52.         condition = " and ".join(["{}=?".format(i) for i in conditions.keys()])
53.         query = "select {} from {} where {}".format(aims, table_name, condition)
54.         self.__cursor.execute(query, list(conditions.values()))
55.         result = self.__cursor.fetchall()
56.         return result
57.
58.     def add_record(self, table_name: str, values: list):
59.         """
60.         添加一条记录
61.         :param table_name: 表名
62.         :param values: 值
63.         """
64.         names = self.__get_columns_name(table_name)
65.         query = "insert into {} ({} ) values ({} )".format(table_name, ','.join(names), ','.join(['?'
        * len(values)])
66.         self.__cursor.execute(query, values)

```

数据库建议语句

```
create index identWaypointsIndex on Waypoints (Ident);  
create index icaoTerminalsIndex on Terminals (ICAO);  
create index terminalidTerminalLegsIndex on TerminalLegs (TerminalID);  
create index icaoAirportsIndex on Airports (ICAO);  
create index airportidRunwaysIndex on Runways (AirportID);  
create index identAirwaysIndex on Airways (Ident);  
create index airwayidAirwayLegsIndex on AirwayLegs (AirwayID);
```