Fenix 数据编写指南 – Eyderoe

2025/2 ver.2 **免费文档，遵守 GPL-3。若付费购买该文档，请寻求退款**

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，Navaids 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。(Sqlite 虽然支持忽略外键约束，但不建议)

后面章节顺序即为建议顺序。

	ID	Ident	Collocated	Name	Latitude	L
1	1	AARON	0	AARON	-79.3845	-1
2	2	ANGIE	0	ANGIE	-89.75466667	-1
3	3	BAKUS	0	BAKUS	-81.43566667	-1
4	4	BAROW	0	BAROW	-80.21816667	-1
5	5	BILLL	0	BILLL	-79.998	
6	6	BLAIR	0	BLAIR	-80.23366667	-1
7	7	BOHIC	0	BOHIC	-81.86516667	-1
8	8	BRIAN	0	BRIAN	-81.89016667	-1
9	9	COREY	0	COREY	-80.18883333	
10	10	DANNY	0	DANNY	-79.3955	
11	11	DEANO	0	DEANO	-79.68283333	-1
12	12	DEBIE	0	DEBIE	-89.95766667	-1
13	13	ELIOT	0	ELIOT	-80.24683333	-1
14	14	ESSPO	0	ESSPO	-79.63366667	-1
15	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在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（仅在导航台上 NAIP 和 AIP 具有差距）即可认为同一。

首先对导航台 Navaids 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，我也不太懂，这里填 0 就行，但应确保 MagneticVariation 磁差中的值准确。

接着则是 NavaidsLookup，这里只说明 NavKeyCode，NavKeyCode 值区别了同区域(ZU, ZP, ZL, …)中相同识别号导航台。

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 Navaids 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(通过无 GS 台信息推测)**

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。

但假如按 424 标准(RW02L,RW02R -> RW02B)写入，注意每个航点需拆开。(建议参照其他多跑道机场程序编写)

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. Wpt 航点显然
e. TurnDir 转向 5.20
f. Nav 推荐导航台
g. Cource 航向 5.26
h. Distance 距离 5.27 假如是 RF 航段，这里是弧段长，而不是弧半径。
i. Alt 高度限制 5.29 5.30 假如是 MAPt 点此处强制填“MAP”
j. WptDescCode 不管，Fenix 应该不会读取这个

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

```
# 使用条件：不会出现分叉
master = ['A', 'C', 'F']
slave = ['X', 'Y', 'A', 'B', 'C']
firstLoc = secondLoc = -1
for i in master:
    if i in slave:
        if firstLoc == -1:
            firstLoc = slave.index(i)
        else:
            secondLoc = slave.index(i)
            break
if firstLoc > secondLoc:
    slave = slave[::-1]

insertLoc = 0
for i in slave:
    if i in master:
        insertLoc = master.index(i) + 1
    else:
        master.insert(insertLoc, i)
        insertLoc += 1
print(master)
```