S3 Data Processing

There are five applications used to gather stellar data and export for use with Star Ship Simulator (S3).

Requirements

- A C++ 20 compiler with CMake support.
- Current versions of the following. I used VCPKG:
    - Spdlog
    - Curlpp
    - Gumbo
    - Libpqxx
    - Sqlite3
- A Postgres database with Postgis installed. I use version v16.
- A disk with 700GB of free space for Gaia CSV data.
- A disk with 140GB of free space for Gaia and Simbad imported data.
- Code access to [Thx1010saascas/S3_Data_Processing (github.com)]

Compiler Setup

I've not used C++ for over 25 years and really got to grips with "make" so forgive me if the setup is not the best! Let me know if anything that can make things easier. The setup is based on CLion.

- Setup the compiler environment so it can download vcpkg packages and install those listed under *Requirements*.
- Add the `ThxSoftLibrary` dependencies. These are generic libraries I created while writing the apps.
    - Create a folder somewhere to store the dependencies. I used **C:\Programming\Libs** but this can be changed as per below.
    - Extract the contents of ThxSoftLibs.zip located at the root of the project source into the new folder.
    - Open CMakeList.txt from the root of the *src* folder and change the `ThxSoftLibrary` path to the folder you created.

That should be it. The information below has example command lines for each app that should be easily adaptable.

Processing

There are five steps in total. Four steps for preparing the data with the final step being the data export to Sqlite.

Each EXE is prefixed with a number to indicate its position in the sequence. Exe parameters will be shown over one or lines for readability.

Step 1. Download Gaia CSV data (1_GaiaDownloadCsvs.exe)

Gaia data is available in a little over 3,600 CSV files. The CSVs we are using contain a superset of the base Gaia data that includes data from other Gaia tables such as metallicity, teff and others. This means that we only need to download this single set of CSV files.

Downloading is slow but concurrent downloading higher than 50 yielded poor results on the occasion I tried it.

Syntax

```
1_GaiaDownloadCsvs
  <Download URL>
  <CSV Storage Folder>
  <Number of Concurrent Downloads>
```

Example

```
1_GaiaDownloadCsvs
  https://cdn.gea.esac.esa.int/Gaia/gdr3/gaia_source/
  D:\Gaia2
  50
```

Step 2: Import Gaia CSVs into Postgres (2_GaiaImportCsvs.EXE)

This step takes a little under nine hours on my mini-PC. It was CPU limited.

Syntax

```
2_GaiaImportCsvs
  <CSV Storage Folder>
  <Postgres Connection String>
  <Maximum Range To Import in Light Years>
  [Number of Concurrent Downloads] Defaults to 5. Max of 15.
```

Example

```
2_GaiaImportCsvs

  "d:\gaia"
  "dbname=star_data user=postgres password=sa hostaddr=127.0.0.1 port=5432"
  10000
  10
```

Step 3: Import Simbad Data (3_SimbadImportData.EXE)

This data is read from Simbad via HTTP and stored directly to the database. This process takes around two hours depending on how Simbad feels at the time.

Syntax

```
3_SimbadImportData
  <Download URL>
  <Postgres Connection String>
```

```
  <Maximum Range To Import in Light Years>
```

Example

```
3_SimbadImportData

"https://simbad.cds.unistra.fr/simbad/sim-tap/sync?REQUEST=doQuery&LANG=ADQL
&FORMAT=csv&QUERY="

  "dbname=star_data user=postgres password=sa hostaddr=127.0.0.1 port=5432"
  10000
```

Step 4: Export Data Fixer (4_ExportDataFixer.EXE)

Many names in Simbad are based on Latin and Greek acronyms. This utility looks at the first three names in the Simbad data and if it finds such a name, adds the names to a row in the **export_overrides** table with an auto generated translation to the full Latin/Greek name.

This table also lets us override a name with any name we choose if necessary.

There also exists a spectral type column to override the spectral type for any stars that we need to tailor.

This utility is non-destructive so re importing Simbad will not invalidate this data.

Entries may be added manually by noting the Simbad **index** value and creating a new row in **export_overrides** with that value.

When step 5 is run, it will use data from this table to override as necessary.

Syntax

```
4_ExportDataFixer
  <Postgres Connection String>
```

Example

```
4_ExportDataFixer
  "dbname=star_data user=postgres password=sa hostaddr=192.168.4.45
port=5432"
```

Step 5: Export S3 Database (5_ExportS3Database.EXE)

The name of the exported file depends on the `[Append Mode]`. If this flag is not equal to “append” then any existing Sqlite file will be deleted and replaced with new content. The name of the file will be based on the `<Export Types>` value. EG: If we range to 2000 ly and set `<Export Types>` to “Cluster,Planet” the generated file name will be “S3_2000ly_ClusterPlanet.db”.

If the `[Append Mode]` equals “append” then the file name will be based only on range. EG: “S3_2000ly.db” and all data. No matter the `<Export Types>` value will be added to that database.

Syntax

```
4_ExportDataFixer
  <Sqlite DB Export Path>
  <Postgres Connection String>
```

```
  <Maximum Range To Export in Light Years>
  <Export Types>
  [Append Mode] (Defaults to false)
```

Example

```
5_ExportS3Database
  "C:\Programming\S3_Data"
  "dbname=star_data user=postgres password=sa hostaddr=192.168.4.45
port=5432"
  2000
  Star,
  Append
```