Anleitung zur Programmfamilie ili2db

Inhaltsverzeichnis

1 Überblick

ilid2db ist eine in Java erstellte Programmfamilie, die zurzeit *ili2pg*, *ili2fgdb*, *ili2gpkg*, *ili2ora*, *ili2mssql*, *ili2mysql* und *ili2h2gis* umfasst.

Damit kann eine INTERLIS-Transferdatei (itf oder xtf) einem INTERLIS-Modell entsprechend (ili) mittels 1:1-Transfer in eine Datenbank (PostgreSQL/Postgis bzw. GeoPackage) gelesen werden oder aus der Datenbank mittels einem 1:1-Transfer eine solche Transferdatei erstellt werden. Folgende Funktionen sind möglich:

- erstellt das Datenbankschema aus einem INTERLIS-Modell
- importiert Daten aus einer Transferdatei in die Datenbank
- exportiert Daten aus der Datenbank in eine Transferdatei

Folgende Transferformate werden unterstützt:

- INTERLIS 1
- INTERLIS 2
- GML 3.2[1]

[1]GML 3.2; die verwendeten Kodierungsregeln entsprechen eCH-0118-1.0

1.1 Schemaimport in die Datenbank

Beim Schemaimport (`--schemaimport`) wird anhand des INTERLIS-Modells das Datenbankschema angelegt.

Diverse Optionen beeinflussen die Abbildung.

Den Geometrien kann mittels Parameter ein EPSG-Code zugewiesen werden. Die Geometrie-Attribute können optional indexiert werden.

Beispiel:

```
java -jar ili2gpkg.jar --schemaimport --dbfile mogis.gpkg path/to/dm01av.ili
```

1.2 Import in die Datenbank

Der Import (`--import`) schreibt alle Objekte (im Sinne der eigentlichen Daten) der Transferdatei in die Datenbank.

Diverse Optionen beeinflussen, was mit den bestehenden Daten in der DB geschieht.

AREA- und SURFACE-Geometrien werden bei INTERLIS 1 polygoniert.

Kreisbögen werden als Kreisbögen importiert und somit nicht segmentiert (oder können optional auch segmentiert werden).

Beispiel:

```
java -jar ili2gpkg.jar --import --dbfile mogis.gpkg path/to/data.xtf
```

1.3 Export aus der Datenbank

Der Export (`--export`) schreibt alle Daten aus der Datenbank in eine Transferdatei.

Mit weiteren Optionen wird gesteuert, welche Daten aus der Datenbank exportiert werden.

Genau einer der Parameter `--models`, `--topics`, `--baskets` oder `--dataset` muss zwingend verwendet werden, um die zu exportierenden DB-Records auszuwählen.

Der Parameter `--exportModels` definiert das Export-Modell, indem die Daten exportiert werden (der Parameter ist also keine Alternative, sondern ein Zusatz für `--models`, `--topics`, `--baskets` oder `--dataset`). Als Export-Modelle sind Basis-Modelle (also z.B. Bundes-Modell statt Kantons-Modell) oder übersetzte Modelle (also z.B. DM_IT statt DM_DE) zulässig. Ohne die Option `--exportModels` werden die Daten so wie sie erfasst sind (bzw. importiert wurden), exportiert.

Geometrien vom Typ *AREA* und *SURFACE* werden bei INTERLIS 1 während dem Export in Linien umgewandelt.

Beispiel:

```

```

```
java -jar ili2gpkg.jar --export --models DM01 --dbfile mogis.gpkg path/
to/output.xtf
```

1.4 Log-Meldungen

Die Log-Meldungen sollen dem Benutzer zeigen, was das Programm macht. Am Anfang erscheinen Angaben zur Programm-Version. Falls das Programm ohne Fehler durchläuft, wird das am Ende ausgegeben.:

```
Info: ili2fgdb-3.10.7-20170823
...
Info: compile models...
...
Info: ...export done
```

Bei einem Fehler wird das am Ende des Programms vermerkt. Der eigentliche Fehler wird aber in der Regel schon früher ausgegeben.:

```
Info: ili2fgdb-3.10.7-20170823
...
Info: compile models...
...
Error: DM01.Bodenbedeckung.BoFlaeche_Geometrie: intersection tids 48, 48
...
Error: ...import failed
```

1.5 Fehlerhafte Daten

Um fehlerhaften Daten zu importieren (um sie danach (z.B. im GIS) zu flicken), muss mindestens die Validierung ausgeschaltet werden (`--disableValidation`). Das DB-Schema muss aber auch so angelegt werden, dass fehlerhafte Werte als Text importiert werden können (`--sqlColsAsText`) bzw. durch `NULL` ersetzt werden können (`--sqlEnableNull`). Und die Programmlogik für den Datenimport muss die Fehler tolerieren (`--skipReferenceErrors` und `--skipGeometryErrors`), so dass z.B. eine Referenz auf ein nicht vorhandenes Objekt ignoriert wird.

Um solche Daten zu importieren (um sie danach zu flicken):

```
java -jar ili2gpkg.jar --schemaimport --sqlEnableNull --sqlColsAsText \
--dbfile mogis.gpkg path/to/mo.ili
```

```
java -jar ili2gpkg.jar --import --skipReferenceErrors --skipGeometryErrors \
--disableValidation --dbfile mogis.gpkg path/to/data.xtf
```

Bei ITF (INTERLIS 1): Fehlerhafte *AREA* Attribute können für den ganzen Datensatz nicht als Polygone gelesen werden, weil ein Programm nicht erkennen kann, welche Linien und Punkte falsch sind (Punkt und/oder Linie zu viel oder zu wenig; Linie zu kurz oder zu lang); und somit

nicht erkennen kann, bei welchem Polygon der Fehler ist. Dass diese Daten nicht gelesen werden können, hat also nicht in erster Linie mit der Validierung zu tun, sondern damit, dass aus den Linien+Punkten keine Polygone gebildet werden können. Die Polygonbildung muss also ausgeschaltet werden (`--skipPolygonBuilding`).

Um solche Daten zu importieren (um sie danach zu flicken):

```
java -jar ili2gpkg.jar --schemaimport --sqlEnableNull --sqlColsAsText \
--skipPolygonBuilding --dbfile mogis.gpkg path/to/mo.ili
```

```
java -jar ili2gpkg.jar --import --skipReferenceErrors --skipPolygonBuilding \
--skipGeometryErrors --disableValidation --dbfile mogis.gpkg path/to/data.itf
```

Bei XTF (INTERLIS 2): Fehlerhafte *SURFACE/AREA* Attribute können als Linien (statt als Polygone) eingelesen werden. Die Polygonbildung muss also ausgeschaltet werden (`--skipPolygonBuilding`).

Um solche Daten zu importieren (um sie danach zu flicken):

```
java -jar ili2gpkg.jar --schemaimport --sqlEnableNull --sqlColsAsText \
--skipPolygonBuilding --dbfile mogis.gpkg path/to/mo.ili
```

```
java -jar ili2gpkg.jar --import --skipReferenceErrors --skipPolygonBuilding \
--skipGeometryErrors --disableValidation --dbfile mogis.gpkg path/to/data.xtf
```

1.6 Laufzeitanforderungen

Das Programm setzt Java 1.8 voraus.

1.6.a PostGIS

Als Datenbank muss mindestens PostgreSQL 8.3 und PostGIS 1.5 vorhanden sein. Falls das INTERLIS-Datenmodell INTERLIS.UUIDOID als OID verwendet, wird die Funktion *uuid_generate_v4()* verwendet. Dazu muss die PostgreSQL-Erweiterung *uuid-ossp* konfiguriert sein (`CREATE EXTENSION "uuid-ossp";`). Mit der Option `--setupPgExt` erstellt *ili2pg* die fehlenden notwendigen Erweiterungen.

1.6.b FileGDB

Es muss Visual Studio 2015 C and C++ Runtimes installiert sein. Je nach Java Version (Die Java Version ist massgebend, nicht die Windows Version) muss die 32-bit oder 64-bit Version dieser Laufzeitbibliothek installiert sein. Falls diese Laufzeitbibliothek nicht installiert ist, gibt es einen Fehler beim laden der FileGDB.dll. Zur Laufzeit entpackt *ili2fgdb* zwei DLLs/Shared-Libraries und lädt diese. Der Benutzer benötigt also die Berechtigungen, um diese Bibliotheken zu laden.

1.6.c GeoPackage

Zur Laufzeit entpackt ili2gpkg eine DLL/Shared-Library und lädt diese. Der Benutzer benötigt also die Berechtigungen, um die Bibliothek zu laden.

1.7 Lizenz

GNU Lesser General Public License

2 Funktionsweise

In den folgenden Abschnitten wird die Funktionsweise anhand einzelner Anwendungsfälle beispielhaft beschrieben. Die detaillierte Beschreibung einzelner Funktionen ist in Abschnitt 3 zu finden.

2.1 Schemaimport-Funktionen

2.1.a Fall 1.1

Die Tabellen existieren nicht und sollen in der Datenbank angelegt werden (`--schemaimport`).

2.1.a.a PostGIS

```
java -jar ili2pg.jar --schemaimport --dbdatabase mogis --dbusr julia \
--dbpwd romeo path/to/dm01.ili
```

Die leeren Tabellen werden im Default-Schema des Benutzers `julia` angelegt. Die Geometrie-Spalten werden in der Tabelle `public.geometry_columns` registriert.

Als Host wird der lokale Rechner angenommen und für die Verbindung zur Datenbank der Standard-Port.

2.1.a.b GeoPackage

```
java -jar ili2gpkg.jar --schemaimport --dbfile mogis.gpkg path/to/dm01.ili
```

Die Geometrie-Spalten werden in den Tabellen `gpkg_contents` und `gpkg_geometry_columns` registriert.

Falls die Datei mogis.gpkg noch nicht existiert, wird sie erzeugt und mit den für GeoPackage nötigen Metatabellen initialisiert. Falls die Datei schon existiert, werden die Tabellen ergänzt.

2.1.a.c FileGDB

```
java -jar ili2fgdb.jar --schemaimport --dbfile mogis.gdb path/to/dm01.ili
```

Falls die Datei mogis.gdb noch nicht existiert, wird sie erzeugt. Falls die Datei schon existiert, werden die Tabellen ergänzt.

Es werden keine Daten importiert, sondern nur die leeren Tabellen angelegt.

2.2 Export-Funktionen
lorem ipsum

2.3 Prüf-Funktionen
lorem ipsum

2.4 Migration von 3.x nach 4.x
lorem ipsum

3 Referenz
In den folgenden Abschnitten werden einzelne Aspekte detailliert, aber isoliert, beschrieben. Die Funktionsweise als Ganzes wird anhand einzelner Anwendungsfälle beispielhaft in Abschnitt 2 (weiter oben) beschrieben.

Die Dokumentation gilt grundsätzlich für alle ili2db-Varianten, ausser es gibt einen spezifischen Hinweis auf PostGIS, GeoPackage oder FileGDB.

3.1 Aufruf-Syntax

3.1.a PostGIS

```
java -jar ili2pg.jar [Options] [file]
```

3.1.b GeoPackage

```
java -jar ili2fgdb.jar [Options] [file]
```

3.1.c FileGDB

```
java -jar ili2fgdb.jar [Options] [file]
```

Der Rückgabewert ist wie folgt:

- 0 Import/Export ok, keine Fehler festgestellt
- !0 Import/Export nicht ok, Fehler festgestellt

Optionen:

Option	Beschreibung
`--import`	Importiert Daten aus einer Transferdatei in die Datenbank. Die Tabellen werden implizit auch angelegt, falls sie noch nicht vorhanden sind (siehe Kapitel Abbildungsregeln). Falls die Tabellen in der Datenbank schon vorhanden sind, können sie zusätzliche Spalten enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben. Falls beim Import ein Datensatz-Identifikator (–dataset) definiert wird, darf dieser Datensatz-Identifikator in der Datenbank noch nicht vorhanden sein. Um die bestehenden Daten zu ersetzen, kann die Option –replace verwendet werden. TODO: Die Tabellen sind schon vorhanden (und entsprechen (nicht) der ili-Klasse).
`--import`	Importiert Daten aus einer Transferdatei in die Datenbank. Die Tabellen werden implizit auch angelegt, falls sie noch nicht vorhanden sind (siehe Kapitel Abbildungsregeln). Falls die Tabellen in der Datenbank schon vorhanden sind, können sie zusätzliche Spalten enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben. Falls beim Import ein Datensatz-Identifikator (–dataset) definiert wird, darf dieser Datensatz-Identifikator in der Datenbank noch nicht vorhanden sein. Um die bestehenden Daten zu ersetzen, kann die Option –replace verwendet werden. TODO: Die Tabellen sind schon vorhanden (und entsprechen (nicht) der ili-Klasse).

Tabelle 1: Optionen

3.2 Optionen

3.2.a `--import`

Importiert Daten aus einer Transferdatei in die Datenbank.

Die Tabellen werden implizit auch angelegt, falls sie noch nicht vorhanden sind (siehe Kapitel Abbildungsregeln). Falls die Tabellen in der Datenbank schon vorhanden sind, können sie zusätzliche Spalten enthalten (z.B. bfsnr, datum etc.), welche beim Import leer bleiben.

Falls beim Import ein Datensatz-Identifikator (–dataset) definiert wird, darf dieser Datensatz-Identifikator in der Datenbank noch nicht vorhanden sein. Um die bestehenden Daten zu ersetzen, kann die Option –replace verwendet werden.

TODO: Die Tabellen sind schon vorhanden (und entsprechen (nicht) der ili-Klasse).

3.2.b `--update`

Aktualisiert die Daten in der Datenbank anhand einer Transferdatei, d.h. neue Objekte werden eingefügt, bestehende Objekte werden aktualisiert und in der Transferdatei nicht mehr vorhandene Objekte werden gelöscht. Diese Funktion bedingt, dass das Datenbankschema mit der Option `--createBasketCol` erstellt wurde, und dass die Klassen und Topics eine stabile OID haben.

lorem ipsum