GSKonverter Dokumentation

Gerhard Schier

Stand 4. Januar 2018

Inhaltsverzeichnis

Teil I.

Einleitung

1. Entwicklungsziele

Der `GSKonverter` ist ein Programm zum Konvertieren von Geodaten.
Die Entwicklungsziele im einzelnen:

Aktualität

`GSKonverter` unterstützt weitgehend die Möglichkeiten aktueller Geodaten-Formate. Komplexe Objektstrukturen (z.B. Multitype-Objekte, verschachtelte Objekte, Objekte mit mehreren Geometrien) bleiben beim Einlesen der Daten erhalten und können entsprechend ausgewertet werden.

Performanz

- Verarbeitung auch sehr großer Dateien bei geringer Speicherauslastung
- Hohe Geschwindigkeit durch parallele Verarbeitung

Transparenz

Es wurde Wert auf aussagekräftige und übersichtlich präsentierte Log-Ausgaben gelegt. Es wird nach Wichtigkeit der Meldung unterschieden (Info, Warnung, Fehler). Die Meldungen werden gruppiert, sodaß gleich lautende Meldungen nur einmal ausgegeben werden. Dies verhindert, daß wichtige Meldungen einfach übersehen werden. Die Meldungs-Details können in einem gesonderten Detailfenster eingesehen werden.

Flexibilität

`GSKonverter` kann mit oder ohne grafische Oberfläche verwendet werden. So kann die Datenkonvertierung auch auf einem Server ohne Fenstermanager stattfinden.
Die Steuerung erfolgt über manuelle Eingabe von Befehlen und ist damit sehr flexibel.

2. Begriffe

Objekt

Ein Objekt (auch “Geoobjekt“, “Feature“) ist die Einheit, mit der der Konverter arbeitet. Objekte werden aus Datenquellen gelesen, möglicherweise gefiltert oder manipuliert und schließlich wieder in ein Datenziel geschrieben.

Jedes Objekt ist einem “Objekttyp“ zugeordnet und enthält i.d.R. eine oder mehrere Geometrien sowie Sachdaten. Objekte können auch Unterobjekte enthalten (geschachtelte Objekte).

Objekttyp

Der Objekttyp legt die Struktur eines Objekts fest. Z.B. enthält der Objekttyp “Flurstück“ bestimmte Objektfelder wie “Flur“, “Gemarkung“, “Flurstücksnummer“ etc. Ein Objekttyp kann weitere Unter-Objekttypen enthalten, z.B. kann der Objekttyp “Flurstück“ einen Objekttyp “Eigentümer“ enthalten. Jedes Objekt ist genau einem Objekttyp zugeordnet.

Objektfeld

Ein Objekttyp enthält Objektfelder. Jedes Objektfeld hat einen Datentyp (Text, Zahl, Datum, Geometrie, usw.). Jeder Wert, der in einem Objekt enthalten ist, gehört zu einem bestimmten Objektfeld.

3. Struktur von Geodaten

Waren Geodaten früher meist sehr einfach aufgebaut (ein Objekt bestand aus 1 Geometrie plus Sachdaten), ist die Zahl der Möglichkeiten spätestens durch die Einführung der "Geography Markup Language" (GML) sehr viel größer geworden. Hier eine Übersicht möglicher Datenstrukturen, die bei der Konvertierung von Geodaten berücksichtigt werden müssen:

3.1. Multitype-Datenquellen

Eine Datenquelle kann nicht nur aus einem Objekttyp bestehen, sondern kann beliebig viele Objekttypen enthalten. Ein Beispiel dafür sind viele GML-Anwendungen (NAS enthält >100 verschiedene Objekttypen).

von `GSKonverter` **unterstützt:** ja

3.2. Schachtelung von Objekttypen

Objekttypen können Unter-Objekttypen enthalten, sodaß eine Baumstruktur von fast beliebiger Schachtelungstiefe entsteht.

von `GSKonverter` **unterstützt:** ja

3.3. Objekttypen mit mehreren Geometrien

Objekttypen können mehrere Geometrien enthalten (z.B. den Umriss als Polygon und den Mittelpunkt als Punkt).

von `GSKonverter` **unterstützt:** ja

3.4. Anonyme und benannte Geometrien

Eine anonyme Geometrie ist eine Geometrie ohne Namen. Klassisches Beispiel sind Shape-Dateien, hier gibt es nur "die Geometrie". GML-Datenquellen können sowohl anonyme Geometrien als auch benannte Geometrie-Objektfelder enthalten.

von `GSKonverter` **unterstützt:** ja

3.5. Nicht-lineare Geometrien

Kreisbögen, Ellipsen, Splines, etc.

von `GSKonverter` **unterstützt:** Kreisbögen bleiben (sofern vom Ausgabeformat unterstützt) bei der Konvertierung erhalten. Andere nicht-lineare Geometrien werden zur Zeit nicht unterstützt.

4. Geodatenformate

4.1. Shape

Ein von der Firma ESRI entwickeltes Geodatenformat. Ist trotz seiner Beschränkungen immer noch sehr wichtig, da es von allen GIS-Programmen unterstützt wird. Das Format besteht eigentlich aus 3 Dateien:

- eine `shp`-Datei, welche die Geometrien enthält.
- eine `shx`-Datei. Enthält einen Index auf die shp-Datei für schnelleren Zugriff auf die Geometrie.
- eine `dbf`-Datei für die Sachdaten.

Es können weitere Dateien vorhanden sein (für Projektion, Zeichensatz, etc.).

Besonderheiten

- Nur lineare Geometrien möglich
- Jedes Objekt kann nur 1 Geometrie haben
- Feldnamen auf 11 Zeichen, Feldwerte auf 255 Zeichen beschränkt
- Jede Koordinate kann zusätzlich zum Rechts- und Hochwert einen z-Wert (Höhe) und einen m-Wert haben

Unterstützung durch `GSKonverter`

- lesen, schreiben (type=shp)
- z- und m-Werte bleiben beim Konvertieren von Shape nach Shape erhalten
- Feldnamen und Feldwerte werden bei Bedarf gekürzt, Feldnamen ausserdem in Großbuchstaben umgewandelt

4.2. Geography Markup Language (GML)

Ein XML-basiertes Geodatenformat. Die Objekttypen werden in Schemadateien (`*.xsd`) definiert. Es gibt viele auf GML aufbauende Fachanwendungen mit jeweils eigenen vordefinierten Objekttypen, z.B.:

- NAS: Austauschformat für Katasterdaten (ATKIS, ALKIS, ...)
- XPlan-GML: Austauschformat für Raumplanung (z.B. Bebauungspläne, Landschaftspläne, ...)

Besonderheiten

Kein anderes Geodatenformat ist vielseitiger als GML. Alle in Kapitel 3 genannten Möglichkeiten werden unterstützt.

Die Komplexität von GML ist dadurch (insbesondere ab Version 3) sehr hoch. Das Verhältnis von Auszeichnungdaten zu den eigentlichen Nutzdaten ist sehr ungünstig. Dadurch erreichen GML-Dateien (z.B. im Verhältnis zu Shape-Dateien) eine enorme Größe, was das Handling der Dateien erschwert. So ist ein Öffnen einer GML-Datei mit einem Texteditor aufgrund der Größe oft unmöglich.

Unterstützung durch `GSKonverter`

- lesen, schreiben (type=gml)
- Die Objektstruktur wird aus der Schemadatei gelesen, dadurch können beliebige GML-Dateien verarbeitet werden.
- Die Objekte werden beim Konvertieren seriell abgearbeitet, dies ermöglicht die Verarbeitung auch sehr großer Dateien.
- GML unterstützt ab Version 3 beliebige Geometrietypen wie Splines etc. `GSKonverter` unterstützt Kreisbögen, sonstige nicht-lineare Geometrien werden nicht unterstützt.
- Die Kodierung der Geometrien bleibt bei Konvertierung von GML nach GML erhalten, solange keine Transformation oder Umprojektion stattfindet. Das betrifft auch nicht unterstützte Geometrietypen.

4.3. Normbasierte Austauschschnittstelle (NAS)

Eine GML-Fachanwendung zum Austausch von Katasterdaten (ALKIS, ATKIS, AFIS) in Deutschland.

Unterstützung durch `GSKonverter`

- lesen (type=nas)
- Die Objektstruktur wird vereinfacht, insbesondere werden wichtige Informationen aus Unterobjekten auf die oberste Objektebene gehoben, z.B. Lebenszeitintervall, Lagebezeichnung u.a.
- Bei Ausgabe einer NAS-Datenquelle als SQL wird eine Beziehungstabelle erzeugt, aus der die Beziehungen zwischen den NAS-Objekten hervorgeht.
- Die NAS-Schemadatei (xsd-Datei) muß über den Befehl `schemalocation` eingebunden werden.

4.4. XPlan-GML

Eine GML-Fachanwendung zum Austausch von Raumplanungs-Vorhaben (z.B. Bebauungspläne, Flächennutzungspläne) in Deutschland.

Unterstützung durch `GSKonverter`

- lesen (type=xplan)
- Die XPlan-Schemadatei (xsd-Datei) muß über den Befehl `schemalocation` eingebunden werden.

4.5. Keyhole Markup Language (KML)

KML ist ebenfalls ein XML-basiertes Geodatenformat. Die Komplexität ist jedoch viel geringer als bei GML. KML wurde von Google entwickelt und ist jetzt ein Standard des *Open Geospatial Consortium*.

Unterstützung durch `GSKonverter`

- lesen, schreiben (type=kml)

4.6. GPS Exchange Format (GPX)

XML-basiertes Geodatenformat zur Speicherung von GPS-Tracks.

Unterstützung durch `GSKonverter`

- nur lesen (type=gpx)

4.7. Postgis

`Postgis` ist eine Erweiterung der Datenbank `PostGres` zur Speicherung und Verarbeitung von räumlichen Daten. `GSKonverter` unterstützt die Ausgabe von sql-Dateien, die dann in eine Postgis-Datenbank importiert werden können.

Unterstützung durch `GSKonverter`

- nur schreiben (type=sql)

4.8. Drawing Interchange Format (DXF)

`DXF` ist ein von der Firma Autodesk entwickeltes Format zum Austausch von CAD-Zeichnungen.

Unterstützung durch `GSKonverter`

- nur schreiben (type=dxf)
- Die DXF-Datei enthält nur die Geometrien, keine Sachdaten.
- Die Geometriefelder werden auf Layer abgebildet, d.h. es wird pro Geometriefeld in der Datenquelle ein Layer in der Ausgabedatei erzeugt.

Teil II.

Bedienung

5. Installation

Die Installation besteht im Entpacken des zip-Archivs.

- Unterverzeichnis `lib` enthält die benötigten Java-Libraries.
- Unterverzeichnis `proj` enthält die proj4-Libraries.
- Unterverzeichnis `pref` wird beim ersten Programmstart angelegt (Schreibrechte vorausgesetzt). Dort werden die Benutzereingaben abgespeichert.

6. Programmstart

Eingabeaufforderung (Konsolenfenster) öffnen und ins Programmverzeichnis wechseln. Der Programmstart mit grafischer Oberfläche erfolgt durch folgenden Befehl:

```
java -jar gskonverter.jar
```

Bei großen Datenmenge kann es nötig sein, den zur Verfügung stehenden Arbeitsspeicher zu vergrößern:

```
java -jar -Xmx500M gskonverter.jar
```

Das Programm kann (z.B. zur Verarbeitung von Geodaten auf einem Server) auch ohne grafische Oberfläche gestartet werden. In diesem Fall müssen die Befehle als Textdatei vorliegen (z.B. befehle.txt). Der Aufruf erfolgt dann wie folgt:

```
java -jar gskonverter.jar befehle.txt
```

7. Grafische Oberfläche

Das wichtigste Element der grafischen Oberfläche ist das Texteingabefeld in der oberen Hälfte. Hier werden die Befehle eingegeben. Bei Klick auf `Start` beginnt die Konvertierung.

Über die Reiter über dem Texteingabefeld kann zwischen verschiedenen Eingabefenstern umgeschaltet werden. Die Eingaben werden beim Beenden gespeichert und stehen beim nächsten Programmstart wieder zur Verfügung. Die Anzahl der Reiter kann bei Bedarf über folgenden Eintrag in der Datei `konverter.properties` im Programmverzeichnes geändert werden (maximal 25 Tabs):

```
numTabs=20
```

In der unteren Hälfte des Fensters werden während der Konvertierung die Tasks aufgelistet. Bei Klick auf einen Task in der Taskliste werden auf der rechten Seite die Meldungen (Fehler, Warnungen, Info) angezeigt, die in diesem Task aufgetreten sind. Die Tasks werden auf Mehrkern-Rechnern parallel abgearbeitet, d.h. es können mehrere Tasks gleichzeitig bearbeitet werden.

8. Befehlseingabe

Die Befehlseingabe erfolgt nach folgendem Muster:

```
befehl wert wert ... [parameter: wert, parameter: wert, ...]
```

Am Zeilenanfang steht immer der Befehlsname (wird **fett** dargestellt). Danach folgen optional einer oder mehrere Eingabewerte. Am Ende der Zeile stehen in eckigen Klammern weitere Parameter.

Beispiel:

```
read *.shp [recursive: true]
```

Leer- und Sonderzeichen

Eingabewerte, die Leerzeichen enthalten, müssen in Anführungszeichen eingeschlossen werden:

```
read "meine punkte.shp"
```

Parameterwerte müssen in Anführungszeichen eingeschlossen werden, wenn sie Leerzeichen, Komma oder Doppelpunkt enthalten.
Es können auch einfache Anführungszeichen verwendet werden, z.B. wenn der Eingabewert ein Anführungszeichen enthält:

```
write [type:csv, textmarker:'"']
```

Teil III.

Referenz

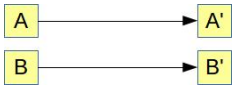
9. Lesen von Geodaten

9.1. Allgemeines

Zum Lesen von Datenquellen können die Befehle `read`, `merge` und `append` verwendet werden.

Alle Befehle können mehrere Dateien verarbeiten. Bei `read` erfolgt die Konvertierung der ausgewählten Datenquellen in voneinander unabhängigen Tasks. Auf Rechnern mit Mehrkernprozessoren ist damit eine parallele Verarbeitung möglich.

```
read A B
```



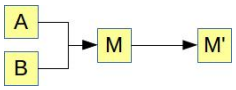
Hier werden zwei voneinander unabhängige Tasks verwendet.

Bei `merge` bzw. `append` werden die ausgewählten Datenquellen nacheinander gelesen und in einen Datenstrom verwandelt, sodaß nur 1 Ausgabedatei entsteht. Der Befehl `append` kann nur nach einem `merge`-Befehl verwendet werden.

```
merge A B
```

ist gleichwertig zu

```
merge A
append B
```



Hier werden die Datenquellen “gemerged“.

Zur Auswahl der Datenquellen kann der Asterisk verwendet werden, z.B.:

```
merge /home/daten/*.shp
```

Es werden alle Dateien mit der Endung “shp“ gelesen.

9.2. Befehl `read`

Bestimmt die zu lesenden Datenquellen. Es können mehrere Datenquellen angegeben werden:

```
read punkte.shp linien.shp flaechen.shp
```

Eine Mehrfachauswahl ist auch mit dem Asterisk (*) möglich:

```
read /home/daten/*.shp
```

Die Verarbeitung der ausgewählten Datenquellen erfolgt parallel.

9.2.1. Parameter

type [string]

Angabe des Datentyps. In der Regel wird der Dateityp an der Dateiendung erkannt. Wenn dies nicht möglich ist, kann der Datentyp explizit angegeben werden.

```
read *.xml [type: gml]
```

recursive [true | false]

Bewirkt rekursive Auswahl der Datenquellen über Unterverzeichnisse.

```
read /home/daten/*.shp [recursive: true]
```

Auswahl aller shp-Dateien in `/home/daten` und den Unterverzeichnissen.

maxSearchLevel [integer]

Wie viele Ebenen von Unterverzeichnissen sollen durchsucht werden. Wird nur ausgewertet, wenn `recursive: true`.

```
read /home/daten/*.shp [recursive: true maxSearchLevel: 2]
```

Es werden 2 Ebenen von Unterverzeichnissen durchsucht.

pattern [string]

Regulärer Suchausdruck, der auf Dateinamen angewendet wird.

```
read /home/daten/*.shp [pattern: flur*]
```

Es werden alle Dateien ausgewählt, deren Name mit “flur“ beginnt.

9.2.2. Parameter für Datentyp `gml`

maxTypeDepth [integer]

Bei GML-Anwendungen (z.B. NAS) kann die Schachtelungstiefe der Objekttypen sehr hoch sein. Dies erhöht die Speicherauslastung erheblich. Da die Unter-Objekttypen i.d.R. nicht benötigt werden, kann die Schachtelungstiefe der Objekttypen beschränkt werden.

```
read /home/daten/*.gml [type: nas, maxTypeDepth: 5]
```

Unter-Objekttypen werden nur bis zur 5.Ebene ausgelesen.

9.2.3. Parameter für Datentyp `csv`

header [true | false]

Gibt an, ob die erste Zeile die Feldnamen enthält.

separator [string]

Angabe des Trennzeichens. Defaultwert ist ";".

encoding [string]

Angabe der Dateikodierung. Wenn nicht definiert, wird die Standardkodierung des Betriebssystems verwendet.

```
read /home/daten/*.txt [encoding: UTF-8]
```

9.3. Befehl `merge`

Fügt die ausgewählten Datenquellen zu einer Datenquelle zusammen.
Mit dem Befehl `append` können weitere Datenquellen angehängt werden.
Mehrfachauswahl ist möglich analog zum Befehl `read`.

9.3.1. Parameter

Die Parameter für den Befehl `read` können auch für `merge` angewendet werden.

method [first | union | intersection]

Methode, mit der die Datenquellen vereinigt werden. Defaultwert ist `first`.

first Maßgeblich ist hier der Objekttyp der ersten gelesenen Datenquelle, dieser wird nicht verändert. Die Objekttypen aller weiteren Datenquellen werden auf diesen Objekttyp abgebildet.

union Es wird ein neuer Objekttyp gebildet, der alle (nicht gleichnamigen) Objektfelder aller Datenquellen enthält.

intersection Es wird ein neuer Objekttyp gebildet, der die Objektfelder enthält, die in allen Datenquellen vorkommen.

```
merge /home/daten/*.shp [method: union]
```

forceMultiType [true | false]

Die Ausgangsdatenquelle ist eine MultiType-Datenquelle mit allen Objekttypen der Eingangsdatenquelle. Die Mergemethode wird automatisch auf `union` gesetzt.
Anwendungsbeispiel: mehrere Shape-Dateien mit unterschiedlichen Objekttypen sollen in 1 GML-Datei geschrieben werden.

```
merge /home/daten/*.shp [forceMultiType: true]
```

9.4. Befehl append

Hängt Datenquellen an eine vorher definierte Merge-Datenquelle an.
Mehrfachauswahl ist möglich analog zum Befehl `read`.

```
merge linien.shp punkte.shp
```

ist gleichwertig zu

```
merge linien.shp
append punkte.shp
```

9.4.1. Parameter

Die Parameter für den Befehl `read` können auch für `append` angewendet werden.

9.5. Kurzeingabe von Geodaten

Mit den Befehlen `read`, `merge` und `append` kann eine Syntax zur "On-the-fly"-Erzeugung von Geodaten verwendet werden.

Die Definition der Features muß in geschweifte Klammern eingeschlossen sein:

```
read { ... }
```

Jedes definierte Feature muß in eckige Klammern eingeschlossen sein, das die (unbenannte) Geometrie sowie optional, wiederum in eckige Klammern eingeschlossen, die Feldwerte (Sachdaten) des Features enthält. Geometrien werden in WKT (Well Known Text) definiert. Eingaben, die Sonderzeichen enthalten, sollten in Anführungszeichen oder Hochkomma eingeschlossen werden.

```
read {[POINT(56.7 12.0)]}
```

Erzeugt ein Feature mit Punktgeometrie ohne Sachdaten.

```
read {[POINT(56.7 12.0) ["Hauptstrasse", 25]]}
```

Erzeugt ein Feature mit Punktgeometrie und zwei automatisch benannten Sachdatenfeldern.

```
read {[POINT(56.7 12.0) [strasse="Hauptstrasse", hausnr=25]]}
```

Erzeugt ein Feature mit Punktgeometrie und zwei Sachdatenfeldern “strasse“ und “hausnr“.

```
read {
        [POINT(56.7 12.0) [strasse="Hauptstrasse", hausnr=25]]
        [POINT(80.9 23.7) [strasse="Dorfweg", hausnr=1]]
        }
```

Erzeugt zwei Features mit Punktgeometrie und Sachdaten.

```
read {
        [line [id: number]]
        ["LINESTRING(56.7 12.0, 88 892.7)" [1005]]
        }
```

Die erste Zeile kann zur expliziten Definition der Feldtypen genutzt werden. Es können folgende Datentypen verwendet werden: `string`, `number`, `integer`, `bool`, `geometry`, `point`, `line`, `polygon`.

```
read {
        [line [id: number, location: point]]
        ["LINESTRING(56.7 12.0, 88 892.7)" [1005, "POINT(33.8 57.1)"]]
        }
```

Erzeugt ein Feature mit unbenannter Liniengeometrie und benannter Punktgeometrie.

10. Schreiben von Geodaten

10.1. Allgemeines

Zum Schreiben der gelesenen Geodaten wird ausschließlich der Befehl `write` verwendet. Als Angabe ist nur der Datentyp, der geschrieben werden soll, zwingend.

```
read ...
write  [type: shp]
```

Die Ausgabe erfolgt im Shape-Format und wird in das Verzeichnis der Quelldatei geschrieben.

Man kann den Ausgabe-Datentyp auch über die Dateiendung angeben:

```
read ...
write *.shp
```

Bei Angabe von relativen Zielverzeichnissen werden diese relativ zum Quellverzeichnis erzeugt:

```
read ...
write ausgabe/*.shp
```

10.1.1. AutoGrouping

Geodaten-Formate können sehr komplex aufgebaut sein (siehe Kapitel 3). Bei der Konvertierung eines komplexen Formates wie z.B. GML in ein Format, das diese Komplexheit nicht unterstützt (z.B. Shape) müssen entsprechend viele Dateien erzeugt werden, um alle Inhalte des Ausgangsformats abzubilden. `GSKonverter` übernimmt dies automatisch in Abhängigkeit vom gewählten Zielformat. Die gelesenen Objekte werden in folgender Reihenfolge gruppiert und dann in die entsprechenden Zieldateien geschrieben:

1. Objekttyp
2. Geometriefeld
3. Geometrytyp (Punkt, Linie, Fläche)
4. Geometrieklasse (einfache oder Multigeometrie)

Die Gruppierung der Objekte erfolgt bis zu dem vom Ausgabeformat verlangten Gruppierungslevel. Die Namen der Ausgabedateien werden durch Anhängen des Objekttypnamens/des Feldnamens/des Geometrietyps/der Geometrieklasse gebildet.

Beispiel: Eine GML-Datei soll nach Shape konvertiert werden. Die Datei enthält die Objekttypen `flurstueck` und `gebaeude`, welche jeweils die Geometriefelder `location` vom Type Polygon und `position` vom Typ Punkt enthalten. Es werden automatisch folgende Shape-Dateien erzeugt:

```
flurstueck-location.shp
flurstueck-position.shp
flurstueck-position-multi.shp
gebaeude-location.shp
gebaeude-position.shp
gebaeude-position-multi.shp
```

Die `*-multi`-Dateien sind notwendig, wenn ein Punktfeld MultiPoints enthalten kann. Das Shape-Format unterstützt nicht Multipoints und einfache Punkte gemischt in 1 Datei.
Der Level der Gruppierung kann auch über den Parameter `groupBy` gesetzt werden. Jedoch kann kein geringerer Level erzwungen werden, als das Ausgabeformat unterstützt.

10.2. Befehl `write`

10.2.1. Parameter

type [string]

Angabe des Datentyps. In der Regel wird der Dateityp an der Dateiendung erkannt. Wenn dies nicht möglich ist, kann der Datentyp explizit angegeben werden.

```
write  [type: gml]
```

groupBy [type | geometryfield | geometrytype | geometryclass]

Bewirkt eine Gruppierung der Ausgabeobjekte nach dem angegebenen Level.

```
read /home/daten/*.gml
write *.sql [groupBy: type]
```

Es wird für jeden in den Ausgangsdaten vorhandenen Objekttyp eine eigene SQL-Datei erzeugt.

```
read /home/daten/*.gml
write *.sql [groupBy: geometryfield]
```

Es wird für jeden in den Ausgangsdaten vorhandenen Objekttyp und zusätzlich für jedes Geometriefeld eine eigene SQL-Datei erzeugt.

10.2.2. Parameter für Datentyp `sql`

schema [string]

Der Name des zu verwendenden Datenbankschemas.

dml [insert | copy]

Abkürzung für "Data Manipulation Language".

insert Für das Einfügen der Daten werden INSERT-Befehle verwendet. Geometrien werden als WKT (Well Known Text) kodiert.

copy Für das Einfügen der Daten werden COPY-Befehle verwendet. Geometrien werden als WKB (Well Known Binary) kodiert.

ddl [create | drop | none]

Abkürzung für "Data Definition Language". Gibt an, ob Befehle zum Erzeugen der Tabellen ausgegeben werden sollen.

create Es werden CREATE TABLE-Befehle ausgegeben.

drop Es werden zusätzlich DROP TABLE-Befehle ausgegeben, d.h. bereits vorhandene Tabellen werden vor dem Einlesen gelöscht.

none Es werden keine DDL-Befehle ausgegeben.

singletransaction [true | false]

Gibt an, ob die SQL-Statements in einer einzigen Transaktion ausgeführt werden.

postgis [1 | 2]

Verwendete PostGis-Version. Die Versionen unterscheiden sich in der Syntax der SQL-Statements.

Folgende Parameter gelten nur für den Datentyp `nas`:

nasrelations [true | false]

Gibt an, ob eine Beziehungstabelle der NAS-Objekte ausgegeben werden soll.

nasfullhistory [true | false]

Nur bei Bestandsdatenaktualisierung: Gibt an, ob verfallene NAS-Objekte in der Datenbank verbleiben sollen.

naslognba [true | false]

Nur bei Bestandsdatenaktualisierung: Gibt an, ob eine Logtabelle geschrieben werden soll, die aktualisierte bzw. gelöschte Objekte enthält.

10.2.3. Parameter für Datentyp `gml`

version [2 | 3]

Angabe der ausgegebenen GML-Version

10.2.4. Parameter für Datentyp `csv`

header [true | false]

Gibt an, ob die erste Zeile die Feldnamen enthält.

geometries [true | false]

Gibt an, ob Geometrien ausgegeben werden. Geometrien werden als WKT (Well Known Text) ausgegeben.

separator [string]

Angabe des Trennzeichens. Defaultwert ist ";".

textmarker [string]

Angabe des Zeichens, mit dem Text gekennzeichnet wird. Defaultwert ist ".

encoding [string]

Angabe der Dateikodierung. Wenn nicht definiert, wird die Standardkodierung des Betriebssystems verwendet.

```
write /home/daten/*.txt [encoding: UTF-8]
```

dateformat [string]

Formatierung von Datumswerten.

```
write /home/daten/*.txt [dateformat:"dd.MM.yyyy HH:mm:ss"]
```

numberformat [string]

Formatierung von Zahlen.

```
write /home/daten/*.txt [numberformat:"#.##0,00"]
```

booleanformat [string]

Formatierung von Ja-/Neinwerten.

```
write /home/daten/*.txt [booleanformat:"ja;nein;?"]
```

coordformat [string]

Formatierung von Koordinaten bei der Ausgabe von Geometrien im WKT-Format.

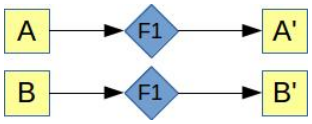
```
write /home/daten/*.txt [coordformat:"#0.000"]
```

11. Filtern von Geodaten

11.1. Allgemeines

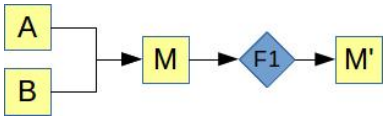
Filter können gelesene Objekt unterdrücken, manipulieren oder sogar aus dem Eingangsobjekt mehrere Ausgangsobjekte erzeugen.
Filter können wahlweise auf alle vorangegangenen Lesebefehle oder durch Voranstellen des Zeichens “>“nur auf den letzten Lesebefehl angewendet werden.
Auf diese Weise können verschieden Datenquellen zu einer Datenquelle zusammengefügt (“gemergt“) und **vor** dem Zusammengefügen gefiltert werden.

```
read A B
filter1
```



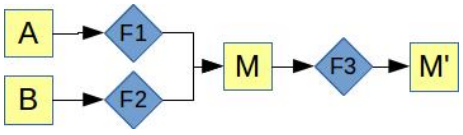
Die Verarbeitung erfolgt in zwei voneinander unabhängigen Tasks, Filter “filter1“ wird auf jeden Task angewendet.

```
merge A B
filter1
```



Die Datenquellen werden gemergt und **danach** gefiltert.

```
merge A
>filter1
append B
>filter2
filter3
```



Die Filter “filter1“ und “filter2“ werden **vor** dem Mergen und nur auf den direkt vorangegangenen Lesebefehl angewendet, während “filter3“ **nach** dem Mergen und damit auf alle Datenquellen angewendet wird.

11.2. Befehl `add`

Fügt neues Objektfeld ein und setzt dessen Inhalt.

Gebrauch:

add *fieldname*
add *fieldname* **as** *datatype*
add *fieldname* **as** *datatype* **set** *expression*

Gültige Datentypen sind: *integer, number, string, bool, date, geometry, point, line, polygon.*

```
add POP_DENS as number set "{POP_CNTRY}/{SQKM_CNTRY}"
```

Füge ein Objektfeld POP_DENS ein und berechne dessen Wert aus dem Inhalt der Felder POP_CNTRY und SQKM_CNTRY.

11.3. Befehl `add_filename`

Fügt den Namen der Ausgangsdatei als neues Objektfeld ein. Der Feldname ist, sofern nicht angegeben, *filename.*

Gebrauch:

add_filename
add_filename [fieldname:*fieldname*]

11.4. Befehl `add_typename`

Fügt den Namen des Objekttyps als neues Objektfeld ein. Der Feldname ist, sofern nicht angegeben, *typename.*

Gebrauch:

add_typename
add_typename [fieldname:*fieldname*]

11.5. Befehl `drop`

Löscht Objektfelder.

Gebrauch:

drop *fieldname*

11.6. Befehl `flatness`

Gibt den Wert an, der bei der Linearisierung von nicht-linearen Geometrien verwendet wird.

Der Wert bezeichnet den maximalen Abstand zwischen Ausgangsgeometrie und linearisierter Geometrie in Geometrieeinheiten.

Je kleiner der Wert ist, desto höher ist die Anzahl der Liniensegmente, in die ein Bogen zerlegt wird.

Gebrauch:

flatness *number*

11.7. Befehl `limit`

Beschränkt die Anzahl der Objekte.

```
limit 5
```

Es werden nur 5 Objekte gelesen.

11.8. Befehl `namespace`

Definiere einen Namespace.

Gebrauch:

namespace *prefix uri*

```
namespace gml http://www.opengis.net/gml/3.2
drop gml:name
```

11.9. Befehl `offset`

Unterdrückt die Objekte bis zur angegebenen Position.

```
offset 100
```

Die ersten 100 Objekte werden unterdrückt.

11.10. Befehl `projection`

Setzt die Projektion der gelesenen Datenquellen.

Gebrauch:

projection [epsg:*value*]
projection *geometryfield* ... [epsg:*value*]

```
projection [epsg:25833]
```

Setzt die Projektion für alle Geometrien.

11.10.1. Parameter

epsg [integer]

EPSG-Code der Projektion

11.11. Befehl `reproject`

Projeziert Geometrien ins angegebene Koordinatensystem.
Wo die aktuelle Projektion, in der die Geometrien vorliegen, nicht bekannt ist, muß diese mit dem Befehl `projection` definiert werden.

Gebrauch:

reproject [epsg:*value*]
reproject *geometryfield* ... [epsg:*value*]

```
reproject [epsg:25833]
```

Projeziert alle Geometrien nach EPSG:25833.

```
reproject location position [epsg:25833]
```

Nur die Geometrien “location“ und “position“ werden umprojeziert.

11.11.1. Parameter

epsg [integer]

EPSG-Code der Projektion

11.12. Befehl `schemalocation`

Definiert den Pfad oder URL zu benötigten XML-Schemadateien. Überschreibt die in der XML-Datenquelle definierten Schemalocations. `schema-id` kann die Schema-URI oder das in der Datenquelle definierte Schema-Prefix sein.

Gebrauch:

schemalocation *schema-id location*
schemalocation *location*

```
schemalocation xplan schemas/XPlanGML_4_0/XPlanung-Operationen.xsd
```

Setze Schemalocation für das Prefix "xplan". Anstelle des Prefix kann man auch die Schema-URI einsetzen.

```
schemalocation schemas/NAS_6.0/schema/NAS-Operationen.xsd
```

Funktioniert dann, wenn in der XML-Datenquelle nur 1 Schemalocation definiert ist.

11.13. Befehl `simpletype`

Macht aus einer Multitype-Datenquelle eine einfache Datenquelle mit anonymer Geometrie. Dies ist nützlich, wenn man einen Objekttyp als Shape ausgeben will und dabei ein bestimmtes Geometriefeld als Geometrie ausgeben will.

Gebrauch:

simpletype *typename*
simpletype *typename/geometryfieldname*

```
simpletype AX_Flurstueck
```

Die gefilterte Datenquelle ist "einfach" und enthält nur noch Objekte des Typs "AX_Flurstueck".

```
simpletype AX_Flurstueck/position
```

Hier wird zusätzlich das Geometriefeld als anonyme Geometrie der Datenquelle verwendet.

11.14. Befehl `simplify`

Vereinfacht Geometrien nach dem Douglas-Peucker-Verfahren.

Gebrauch:

simplify [tolerance:*value*]
simplify *geometryfield* ... [tolerance:*value*]

11.14.1. Parameter

tolerance [float]

Je größer der Toleranzwert, desto stärker wird die Geometrie vereinfacht.

11.15. Befehl `transform`

Transformiert Geometrien mit Hilfe einer affinen Transformation. Ermöglicht Verschieben, Skalieren, Drehen. Optional können Geometriefelder definiert werden, wenn nur bestimmte Geometrien transformiert werden sollen. Wenn nicht angegeben, werden alle Geometrien transformiert.

Gebrauch:

transform [*param*:value, ...]
transform *geometryfield* ... [*param*:value, ...]

11.15.1. Parameter

tx [float]

Verschiebung in x-Richtung

ty [float]

Verschiebung in y-Richtung

scx [float]

Skalierung in x-Richtung

scy [float]

Skalierung in y-Richtung

shx [float]

Scherung in x-Richtung

shy [float]

Scherung in y-Richtung

11.16. Befehl `types`

Filtert Multitype-Datenquellen nach Objekttypen.
Bei GML-Datenquellen kann auch nach (abstrakten) Elterntypen gefiltert werden (Beispiel: `AX_TatsaechlicheNutzung` bei Datentyp `nas`).

Gebrauch:

types *typename ...*
types *typename ...* [exclude:true]

```
types AX_Flurstueck AX_Gebaude
```

Es werden nur Objekte vom Typ “AX_Flurstueck“ und “AX_Gebaeude“ durchgelassen.

```
types AX_Flurstueck AX_Gebaude [exclude:true]
```

Objekte vom Typ “AX_Flurstueck“ und “AX_Gebaeude“ werden unterdrückt, alle anderen werden durchgelassen.

11.16.1. Parameter

exclude [true | false]

Gibt an, ob die Objekttypen unterdrückt werden sollen.

11.17. Befehl `where`

Filtert Objekte mit Hilfe von Ausdrücken.

Gebrauch:

where *ausdruck*

Feldnamen werden in Ausdrücken durch geschweifte Klammern gekennzeichnet, z.B. {*feldname*}.
Strings innerhalb eines Ausdrucks müssen in einfachen Anführungszeichen stehen.

```
where "{CNTRY_NAME} = 'Germany'"
```

Es werden nur die Objekte durchgelassen, deren Objektfeld “CNTRY_NAME”’den Wert “Germany“ hat.