Konferenz der kantonalen Gesundheitsdirektorinnen und -direktoren
Conférence des directrices et directeurs cantonaux de la santé
Conferenza delle direttrici e dei direttori cantonali della sanità

DIE SPITÄLER DER SCHWEIZ
LES HÔPITAUX DE SUISSE
GLI OSPEDALI SVIZZERI

SPLG-Grouper 2023

Integrationshandbuch

Gesundheitsdirektion Kanton Zürich
Amt für Gesundheit
Stampfenbachstrasse 30, Postfach
8090 Zürich
splg@gd.zh.ch
31.10.2022 09:41

Inhalt

Überblick 3
Gruppieren 3
Service 4
Klassen 7
splg.grouper.Grouper 7
splg.data.falldaten.Falldaten 7
splg.output.grouper.GrouperOutput 9
splg.definition.spitalliste.Spitalliste 10
splg.output.gaf.GafBuilder 10
splg.definition.splgcatalog.SplgCatalog 10

Überblick

Dieses Dokument beschreibt, wie der SPLG-Grouper in Softwareprodukte der IT-Partner integriert werden kann. Da der SPLG-Grouper in Java geschrieben ist, funktioniert die Integration am einfachsten in Java-basierte Software. Falls das Softwareprodukt mit anderen Technologien realisiert wurde (beispielsweise C#/.NET), muss gegebenenfalls ein Übersetzungslayer erstellt oder die Socketschnittstelle verwendet werden.

Im Folgenden wird zuerst erklärt, wie man den SPLG-Grouper instanziiert und verwendet. Danach wird als Alternative gezeigt, wie man mit dem socketbasierten Service arbeitet. Und zu guter Letzt werden die benötigten Klassen dokumentiert.

Gruppieren

Eine Instanz des Groupers kann mit Grouper.create() erstellt werden. Der Grouper besitzt Ressourcen, die wieder freigegeben werden müssen. Daher sollte nach Abschluss der Arbeiten die close-Methode des Grouper-Objekts aufgerufen werden. Dies kann explizit passieren, oder implizit bei Verwendung des try-finally-Konstrukts.

```
Bereich bereich = Bereich.Akutsomatik;
File defdir = ... // Pfad auf das Definitions-Verzeichnis (z.B. 4_lib/Akut/2022)
try (Grouper grouper = Grouper.create(bereich, defdir, "")) {
    // Vorbereiten der Eingabedaten (Falldaten)
    Falldaten fall = ...

    // Durchführen der Gruppierung
    GrouperOutput output = grouper.group(falldaten);

    // Weiterverwenden des Grouper-Outputs
    System.out.println(output.toXML());
}
```

Sollen für den gruppierten Fall die Informationen der GAF-Datei erzeugt werden, so geschieht das wie folgt:

```
File catfile = ... // Pfad auf die Katalogdatei (z.B.
                   // 4_lib/Akut/2022/catalog_akutsomatik_2021.1.0.dat)
SplgCatalog catalog = SplgCatalog.load(catfile);
GafBuilder gaf = GafBuilder.getBasicBuilder(catalog);
System.out.println("GAF:");
System.out.println(gaf.getHeader());
for (String line : gaf.getLines("TST", falldaten, output)) {
    System.out.println(line);
}
```

Damit der Grouper funktioniert, muss eine gültige Lizenzdatei vorhanden sein. Diese kann entweder im Definitions-Verzeichnis (z.B. 4_lib/Akut/2022) oder im übergeordneten Verzeichnis (also 4_lib/Akut) oder im Basisverzeichnis (also 4_lib) abgelegt werden.

Die Flexibilität bezüglich der Ablage der Lizenzdatei ist hilfreich, wenn zum Beispiel für verschiedene Bereiche (Akut und Reha) verschiedene Lizenzen zum Einsatz kommen sollen. Der Nachteil ist, dass man im Fehlerfall alle drei Orte prüfen muss.

Wir empfehlen die Lizenz immer direkt im Basisverzeichnis abzulegen (also 4_lib).

Ist die Lizenzdatei nicht vorhanden, ungültig oder enthält sie nicht die nötigen Module (z. B. Module Grouper_Akutsomatik für den Bereich «Akutsomatik»), so wird beim Aufruf von Grouper.create eine IllegalStateException geworfen.

Der Name der Lizenzdatei sollte wie geliefert beibehalten werden. Pflicht ist aber nur, dass der Name mit «License» beginnt und mit «.txt» endet. Gross- und Kleinschreibung ist nicht relevant.

Service

Der SPLG-Grouper kann als Service gestartet werden. Damit wird eine einfache, socketbasierte Schnittstelle geboten, die unabhängig von Java ist. Dazu muss der Service mit dem Aufruf

```
java -cp splg.grouper.jar splg.service.SPLGGrouperService [argumente]
```

gestartet werden. Folgende Parameter können mitgegeben werden:

--libdir [pfad]	Pfad auf das Lib-Verzeichnis (z. B. 4_lib)
--akut	Aktiviert den Bereich Akutsomatik
--reha	Aktiviert den Bereich Rehabilitation
--psy	Aktiviert den Bereich Psychiatrie
--defversion [version]	Legt die zu verwendende Defversion fest (z. B. 2022)
--hostname [hostname]	Der Hostname, auf den der Service gebunden werden soll, typischerweise «localhost».
--port [port]	Der port, unter dem der Service verfügbar sein soll, default ist 6969.
--silent	Legt fest, dass der Service nichts auf Standard-Output schreiben soll.

Der Service implementiert ein einfaches Abfrageprotokoll. Dabei sendet der Client Anfragen und der Service beantwortet sie. Der Service ist single-threaded und kann nur einen Client zur selben Zeit bedienen.

Die Konfigurationsdateien müssen inklusive Bereich abgelegt werden. Das heisst, die Defversion 2022 in der Akutsomatik wäre zu finden in 4_lib/Akut/2022, wobei 4_lib das konfigurierte Lib-Verzeichnis ist. Analog wäre die Defversion 2022 der Rehabilitation zu finden in 4_lib/Reha/2022. Damit können mit derselben Installation Daten für alle Bereiche gruppiert werden.

Jede Anfrage besteht aus mehreren Zeilen. Die erste identifiziert die Art der Anfrage. Die folgenden Zeilen sind optional und enthalten – falls vorhanden – die nötigen Informationen für die Anfrage (beispielsweise die Falldaten). Diese Zeilen müssen alle mit dem Zeichen + beginnen. Der Service entfernt das + beim Lesen der Zeilen. Die letzte Zeile ist immer leer. Damit erkennt der Service, dass die Anfrage fertig ist.

Die Antwort des Service besteht ebenfalls aus mehreren Zeilen. Die erste Zeile beschreibt die Antwort. Die folgenden Zeilen sind optional und enthalten – falls vorhanden – die Nutzdaten der Antwort (beispielsweise das Gruppierungsresultat. Diese Zeilen beginnen mit dem Zeichen +. Bei der Auswertung der Antwortdaten muss das +-Zeichen entfernt werden. Die letzte Zeile der Antwort ist immer leer. Damit erkennt der Client, dass die Antwort fertig ist und der Service für die nächste Anfrage bereit ist.

Folgende Anfrage-Typen sind verfügbar:

BEREICH	Setzt den Bereich. Gültige Werte sind ‘Akutsomatik’, ‘Rehabilitation’ und ‘Psychiatrie’.
DEFVERSION	Setzt die Defversion. Die gültigen Werte hängen von der Installation ab. Typische Werte sind ‘2022’, ‘2021’, ‘2020’.
OUTPUTFORMAT	Setzt das Ausgabeformat. Gültige Werte sind ‘TEXT’, ‘XML’ und ‘JSON’.
GRUPPIEREN	Gruppiert einen Fall. Die Nutzdaten enthalten die Falldaten in einem unterstützten Format (BFS-MS, PRISMA, SPLG-TEXT usw.)
BEENDEN	Beendet die aktuelle Verbindung. Der Service läuft weiter und der Client kann später wieder verbinden.
HERUNTERFAHREN	Beendet den Service. Damit der Client wieder verbinden kann, muss der Service neu gestartet werden.

Sowohl Anfragen als auch Antworten sind in UTF-8 Text kodiert. Zeilenenden sind wahlweise ‘\r\n’ oder ‘\n’.

Beispielabfrage:

```
BEREICH
+Akutsomatik

BEREICH OK - Akutsomatik

DEFVERSION
+2021

DEFVERSION OK - 2021

OUTPUTFORMAT
+XML

OUTPUTFORMAT OK - XML

GRUPPIEREN
+SPLG-INPUT
+bur=12345678;plz=8000;wkt=ZH;fallid=1234;agey=62;aged=0;sfall=A;behart=3;soastat=1
+ICD C541;K660
+CHOP 6861:::2021041915[[7601000000000:1],[7601000000001:4]];6541:0::20210419

GRUPPIEREN OK
+<grouperoutput>
+ <fallid>1234</fallid>
+ <splg>GYNT</splg>
+ <stsl>9</stsl>
+ <lactrl>99</lactrl>
+ <lactrlcodes>[]</lactrlcodes>
+ <mfzs>GYNT</mfzs>
+ <mfzo>GYNT</mfzo>
+ <mfzopunkte gln="7601000000000 lg="GYNT">0.5</mfzopunkte>
+ <mfzopunkte gln="7601000000001 lg="GYNT">0.5</mfzopunkte>
+ <notes></notes>
+ <errors>
+  <error code="0">Fehlerfrei</error>
+ </errors>
+</grouperoutput>

BEENDEN
```

```
Process finished with exit code 0
```

Schwarz sind Anfragen des Clients, grün sind Antworten des Service.

Klassen

splg.grouper.Grouper

Die Grouper-Klasse implementiert den eigentlichen Gruppierungsalgorithmus. Nachdem die Instanz über die statische Methode create() erzeugt wurde, kann mit der Methode group() Fall für Fall gruppiert werden. Nach Abschluss der Gruppierungen sollte die Methode close() aufgerufen werden. Das führt zum Schreiben der Ausgabedateien und gibt Ressourcen frei.

Die Getter-Methoden bieten einfache Infos zu den verfügbaren SPLG, MFZS und MFZO respektive den Spitälern und Standorten der Spitallisten.

Können die Definitionsdateien nicht geladen werden, so wirft die Grouper.create() Methode eine IOException.

Ist die Lizenzdatei nicht vorhanden, nicht gültig oder enthält sie nicht die nötigen Module für den gewählten Bereich, so wirft die Grouper.create() Methode eine IllegalStateException. Ebenso wird bei jedem Aufruf von group() die Lizenz überprüft und ggf. eine IllegalStateException geworfen.

```
public class Grouper implements AutoCloseable {
    public static Grouper create(Bereich bereich,
                                 File defdir,
                                 String kanton) throws IOException;

    public GrouperOutput group(Falldaten fall);

    public List<String> getLGs();
    public List<String> getMFZSs();
    public List<String> getMFZOs();
    public Spitalliste getLeistungsauftraege();

    public void close() throws Exception;
}

public enum Bereich {
    Akutsomatik,
    Rehabilitation,
    Psychiatrie
}
```

splg.data.falldaten.Falldaten

Die Falldaten-Klasse setzt das abstrakte Datenformat um. Dieses ist im Anwenderhandbuch dokumentiert. Alle Fälle werden beim Einlesen der konkreten Eingabeformate in Falldaten-Instanzen abgefüllt. Software, die den SPLG-Grouper integriert, füllt direkt die Falldaten-Instanz ab.

```
public class Falldaten {
    // controlling-relevant
    public String burnr = "";
    public String plz = "";
    public String standort = "";
    public String wohnkanton = "";
    public String statistikfall = "";
    public String behandlungsart = "";
    public String tarifsystem = "";
```

```
    // fallidentifikation
    public String fallid = "";

    // grouper-relevant
    public String agey = "";
    public String aged = "";
    public String ssw = "";
    public String ggw = "";
    public String dmb = "";
    public String freiwilligkeit = "";
    public String austritt = ""; // Austrittsdatum, Format YYYYMMDD

    // grouper-relevant
    public List<FalldatenDiagnose> diagnosen = new ArrayList<>();
    public List<FalldatenBehandlung> behandlungen = new ArrayList<>();

    // output-relevant
    public String ave = "";
    public String weg = "";
    public String sn = "";
    public String skz = "";
    public String ea = "";
    public String ad = "";
    public String ana = "";
    public String ed = "";
    public String mk = "";
    public String ei = "";
    public String gew = "";
    public String hktr = "";

    // output-relevant SwissDRG/TARPSY/STReha
    public String drg = "";
    public String pcg = "";
    public String rcg = "";
    public String mdc = "";
    public String cw = "";
    public String pccl = "";
    public String ecwt = "";

    // zusatz (durchreiche)
    public String zusatz = "";
    public String erhebungsjahr = "";
    public String standortkanton = "";
}

public class FalldatenDiagnose {
    public int rang;
    public String code;
    public String seitigkeit = "";
    public String zusatz = "";
}

public class FalldatenBehandlung {
    public int rang;
    public String code;
    public String seitigkeit = "";
    public String ambExt = "";
```

```
    public String beginn = "";
    public List<FalldatenOperateur> operateure = null;
}

public class FalldatenOperateur {
    public String gln = "";
    public String funktion = "";  // 1 = Erstoperateur, 2 = Zweitoperateur
    public String zulassung = ""; // 1 = Auf Liste GD,  0 = Nicht auf Liste GD
}
```

splg.output.grouper.GrouperOutput

Die Resultate des Gruppierungsvorganges werden als Instanz der Klasse GrouperOutput zurückgegeben. Die drei Methoden toString(), toXML() und toJSON() formatieren die Daten in den drei entsprechenden Formaten. Die übrigen Methoden erlauben Zugriff auf die Daten.

```
public class GrouperOutput {
    public String getFallid() {}
    public String getSplg() {}
    public List<String> getAllSplgs() {}
    public int getStsl() {}
    public int getLactrl() {}
    public String getLactrlCode() {}
    public List<String> getMfz() {}
    public List<MfzOperateur> getMfzoPunkte() {}
    public List<String> getMfzo() {}
    public List<String> getQuer() {}
    public String getNotes() {}
    public String getZusatz() {}
    public ErrorCode getErrorcode() {}
    public List<ErrorCode> getAllErrorcodes() {}
    public String getMfzoString() {}
    public String toString() {}
    public String toXML() {}
    public String toJSON() {}
}

public class MfzOperateur {
    public String getGln() {}
    public List<LgPunkt> getSplgpunkte() {}
    public String toString() {}
}

public class LgPunkt {
    public String getLg() {}
    public double getPunkte() {}
    public String toString() {}
}

public class ErrorCode implements Comparable<ErrorCode> {
    public String getCode() {}
    public String getText() {}
    public int getPriority() {}
    public boolean isOK() {}
    public boolean isWarning() {}
    public boolean isError() {}
    public String toString() {}
}
```

splg.definition.spitalliste.Spitalliste

Die Klasse Spitalliste wird intern vom Grouper verwendet. Nach aussen sichtbar sind zwei Methoden, die die Liste der Spitäler und die Liste der Standorte zurückgeben. Dabei werden die Spitäler über ihre BUR-Nummer und die Standorte über BUR-Nummer und Postleitzahl, respektive BUR-Nummer, Postleitzahl und Standortnummer (getrennt mit '_') geführt.

```
public class Spitalliste {
    public List<String> getSpitaeler() {}
    public List<String> getStandorte() {}
}
```

splg.output.gaf.GafBuilder

Die abstrakte Klasse GafBuilder erlaubt die Konstruktion von einfachen und ausführlichen GafBuilder-Instanzen, welche dazu dienen, die GAF-Datei zu schreiben.

Die Methode getHeader() liefert die Kopfzeile der GAF-Datei.

Die Methode getLines() liefert ein Array von Zeilen für den gegebenen Fall. Diese Zeilen werden 1:1 ins GAF geschrieben.

```
public abstract class GafBuilder {
    public abstract String getHeader();
    public abstract String[] getLines(String skz, Falldaten fall,
                                      GrouperOutput output);

    public static GafBuilder getBasicBuilder(SplgCatalog catalog) {}
    public static GafBuilder getVerboseBuilder(SplgCatalog catalog) {}
}
```

splg.definition.splgcatalog.SplgCatalog

Die Catalog-Klasse bietet Zugriff auf die Informationen der Katalogdateien (z. B. catalog_akutsomatik_2023.1.0.dat). Der GafBuilder verwendet eine Catalog-Instanz, um das GAF zu schreiben. Der Grouper benötigt hingegen keinen Catalog.

```
public class SplgCatalog implements Iterable<CatalogEntry> {
    public static final CatalogType ICD = CatalogType.ICD;
    public static final CatalogType CHOP = CatalogType.CHOP;

    public static SplgCatalog load(File file) throws IOException {}
    public Iterator<CatalogEntry> iterator() {}
    public CatalogEntry getEntry(CatalogType type, String code) {}
}

public class CatalogEntry {
    public CatalogType type;
    public String length;
    public String fullCode;
    public String abbrevCode;
    public String codable;
    public String textDE;
    public String textFR;
    public String nnb = "";
```

```
    public String child = "";
    public String avs = "";

    public List<String> splg = new ArrayList<>();

    public String toString() {}
    public String display() {}
}
```