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 9

For the year 2020

Manual for technical integration

Copyright © 2019 Gesundheitsdirektion Kanton Zürich
Stampfenbachstrasse 30
8090 Zürich
splg@gd.zh.ch

Table of contents

Overview 3
Group 3
Service 4
Classes 6
splg.grouper.Grouper 6
splg.data.falldaten.Falldaten 6
splg.output.grouper.GrouperOutput 7
splg.definition.leistungsauftraege.Leistungsauftraege 8
splg.output.gaf.GafBuilder 8
splg.catalog.Catalog 9

Overview

This document describes how the SPLG-Grouper can be integrated into third-party software. Since the SPLG-Grouper is written in Java, the integration process works best for Java-based software. If the software is based on a different technology (e.g. C#/.NET), the developer may need to create a translation layer or use the socket-based interface.

First, we explain how you can instantiate and use the SPLG-Grouper. Then, we propose an alternative solution using the socket-based interface. Lastly, we document the provided classes.

Group

An instance of the SPLG-Grouper can be created by calling Grouper.create(). The grouper allocates resources, which should be released. Therefore, when you do not use the grouper instance anymore, the close method should be called. This can be done explicitely or implicitely by using the try-finally construct of Java.

```
Bereich bereich = Bereich.Akutsomatik;
File defdir = ... // Path to the definitions (e.g. 4_lib/Akut/2019)
try (Grouper grouper = Grouper.create(bereich, defdir, "")) {
    // Preparation of input (case data)
    Falldaten fall = ...

    // Group the data
    GrouperOutput output = grouper.group(falldaten);

    // Further processing of the grouper output
    System.out.println(output.toXML());
}
```

If you need the information for an input case as is contained in the GAF file, you can do the following:

```
Catalog catalog = new Catalog(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);
}
```

In this code, “catfile” refers to the catalog file in the definition folder (e.g. 4_lib/Akut/2019/Cat2019.1.0.dat).

In order for the grouper to work correctly, you need to supply a valid license file. This license file can be stored either in the definitions folder (e.g. 4_lib/Akut/2019), in its parent folder (e.g. 4_lib/Akut) or in the base folder (e.g. 4_lib).

The multiple possible locations for the license file are helpful if you want to use separate license files for e.g. Akutsomatik and Rehabilitation. The downside is possible confusion.

We recommend to store the license file in the base folder (e.g. 4_lib).

If the license file is not found, invalid or does not contain the required modules (e.g. module Grouper_Akutsomatik for Bereich “Akutsomatik”), the call to Grouper.create will throw an instance of IllegalStateException.

The name of the license file should be retains as is. But it is only required to start with “License” and end with “.txt”. Upper and lower case is ignored.

Service

The SPLG-Grouper can be started as a service. This provides a simple socket based interface, which is independent from the Java technology. For this, start the service using the following call:

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

The following parameters can be provided:

--libdir [path]	Path to the lib folder (e.g. 4_lib)
--akut	Activates Akutsomatik
--reha	Activates Rehabilitation
--psy	Activates Psychiatrie
--defversion [version]	Defines, which Defversion to use (e.g. 2019)
--hostname [hostname]	The hostname to bind the service to, typically «localhost».
--port [port]	The port to bind the service to, default is 6969.
--silent	Activates silent mode, i.e. the service will not write to standard output.

The service implements a simple request-response protocol. The client sends a request and the service responds. The service is single threaded and can only serve one client at a time.

The configuration files have to be stored correctly. For example, the Defversion 2019 in Akutsomatik needs to be stored in 4_lib/Akut/2019, where 4_lib signifies the lib folder as configured. In analogy, the Defversion 2020 for Rehabilitation is stored in 4_lib/Reha/2020. This scheme allows to group data for Akutsomatik and Rehabilitation with only one grouper installation.

Each request is composed of multiple lines. The first line identifies the request type. The lines after the first one are optional and contain – if provided – the required information for the request (e.g. the case data). These lines must start with the character +. The service removes the + character when reading the lines. The last line must be empty. This allows the service to detect the end of the request.

The response consists also of multiple lines. The first line describes the response. The lines after the first one are optional and contain – if provided – the payload of the response (e.g. the result of the grouping process). These lines start with the character +. When reading the response, you have to strip these + characters. The last line of the response is always empty. This allows the client to detect the end of the response and signifies that the service is ready for the next request.

The following request types are available:

BEREICH	Set the “Bereich”. Valid values are ‘Akutsomatik’, ‘Rehabilitation’ and ‘Psychiatrie’.
DEFVERSION	Set the Defversion. Valid values depend on the installation. Typical values would be ‘2020’, ‘2019’, ‘2018’.
OUTPUTFORMAT	Set the output format. Valid values are ‘TEXT’, ‘XML’ and ‘JSON’.

GRUPPIEREN	Groups a case. The case has to be passed in a supported format (e.g. BFS-MS, PRISMA, SPLG-TEXT etc.)
BEENDEN	Terminates the current connection. The service continues to run and the client (or another client) can reconnect later on.
HERUNTERFAHREN	Terminates the service. In order for the client to reconnect, the service must first be started again.

Requests and responses are encoded using UTF-8. Line endings are either '\r\n' or '\n'.

Example request/response:

```
BEREICH
+Akutsomatik

BEREICH OK - Akutsomatik

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:::2018041915[[7601000000000:1],[7601000000001:4]];6541:0::20180419

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
```

The requests are written in black, the responses are written in green.

Classes

splg.grouper.Grouper

The class Grouper implements the core grouping algorithm. After creating an instance using the static method create(), you can group a case using the method group(). After all cases are grouped, you have to call the method close(). This writes the output files and releases any resources.

The getter methods provide information about available SPLG, MFZS, MFZO, hospital names and locations according to the spitallisten.

If the definition files cannot be loaded, the call to Grouper.create() throws an IOException.

If the license file is not found, invalid or does not contain the required Modules for the selected Bereich, then the Grouper.create() call throws an IllegalStateException. Furthermore, with each call to the group() method, the license is checked and – if not valid – an IllegalStateException is thrown.

```
public class Grouper implements AutoCloseable {
    public static Grouper create(Bereich bereich, File defdir, String kt);
    public GrouperOutput group(Falldaten fall);
    public List<String> getLGs();
    public List<String> getMFZSs();
    public List<String> getMFZOs();
    public Leistungsauftraege getLeistungsauftraege();
    public void close();
}

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

splg.data.falldaten.Falldaten

The class Falldaten implements the abstract data format. This format is described in the user manual. On reading the concreate input formats, all cases are stored in Falldaten instances. Software solutions that integrate the SPLG-Grouper do this directly.

```
public class Falldaten {
    // fallidentifikation
    public String fallid = "";

    // grouper-relevant
    public String agey = "";
    public String aged = "";
    public String ssw = "";
    public String ggw = "";
    public String dmb = "";
    public List<FalldatenDiagnose> diagnosen = new ArrayList<>();
    public List<FalldatenBehandlung> behandlungen = new ArrayList<>();

    // controlling-relevant
    public String burnr = "";
    public String plz = "";
```

```
    public String wohnkanton = "";
    public String statistikfall = "";
    public String behandlungsart = "";
    public String tarifsystem = "";

    // output-relevant
    public String ave = "";
    public String weg = "";
    public String pccl = "";
    public String sn = "";
    public String skz = "";
    public String drg = "";
    public String cw = "";
    public String mdc = "";
    public String ea = "";
    public String ad = "";
    public String ana = "";
    public String ed = "";
    public String mk = "";
    public String ei = "";
    public String gew = "";
}

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 String op1 = "";
    public String rolle1;
    public String op2 = "";
    public String rolle2;
}
```

splg.output.grouper.GrouperOutput

The results of the grouping process are returned as instances of class GrouperOutput. The three methods toString(), toXML() and toJSON() format the data in the corresponding formats. The other methods allow access to the data directly.

```
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 String getNotes() {}
    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.leistungsauftraege.Leistungsauftraege

The class Leistungsauftraege is used internally by the grouper. It contains the spitallisten. To the public, there are two methods available, which return the list of hospitals and the list of hospital locations. The hospitals are identified by their BUR number and the hospital locations by the combination of BUR number and postal code (separated by '_').

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

splg.output.gaf.GafBuilder

The abstract class GafBuilder allows instantiation of simple and verbose GafBuilder instances. These are used to write GAF files.

The method getHeader() returns the header line of the GAF file.

The method getLines() returns an array of lines for the given case. These lines are written verbatim into the GAF file.

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

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

splg.catalog.Catalog

The class Catalog provides access to the data contained in the catalog files (e.g. Cat2019.1.0.dat). The GafBuilder uses an instance of Catalog to write the GAF file. The grouper itself does not need Catalog instances.

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

    public Catalog(File catfile) throws IOException {}
    public Iterator<CatalogEntry> iterator() {}
    public CatalogEntry getEntry(CatalogType type, String code) {}
    public static Catalog loadFromDirectory(File directory)
                                            throws IOException {}
}

public class CatalogEntry {
    public String toString() {}
    public String display() {}

    public CatalogType getType() {}
    public String getLength() {}
    public String getFullCode() {}
    public String getAbbrevCode() {}
    public String getCodable() {}
    public String getText() {}
    public String getNnb() {}
    public String getChild() {}
    public String getAvs() {}
    public List<String> getSplg() {}
}
```