JUSTIZ
JUSTIZRESSORT
Bundesministerium für Justiz

Elektronischer Rechtsverkehr (ERV)

Spezifikation des Validierungsmoduls für StaEingabe

Dateiname: STA_Spezifikation_Validierungsmodul.doc

Version: 1.0.0 vom 21.04.2016

Ersteller: Norbert Reinwald, Reinhard Wagner

1 Dokumentinformation

1.1 Inhaltsverzeichnis

1.2 Tabellenverzeichnis

1.3 Änderungsverlauf

Version	Datum	Ersteller	Kommentar
1.0.0	21.04.2016	Reinwald / Wagner	Initialversion

2 Einleitung

2.1 Zweck des Dokuments

Dieses Dokument enthält die Beschreibung des Validierungsmoduls für Eingaben an Staatsanwaltschaften im Rahmen des ERV.

Das Validierungsmodul wird sowohl bei den Übermittlungsstellen als auch in der BRZG zur Prüfung von sämtlichen Eingaben an die Justizanwendung mit der Kennung STA eingesetzt.

Es werden die APIs sowie auch die externen Abhängigkeiten beschrieben, sodass eine problemlose Einbindung des Validierungsmoduls in die Umgebung der Übermittlungsstelle möglich ist.

3 Validierungsmodul

3.1 Deployment Units

Das Validierungsmodul ist als Java-Programm realisiert. Die Validierung (für applikationsspezifische Prüfungen) besteht aus einer Validierungsklasse, einigen Hilfsklassen und den dazu benötigten Ressourcen, die in einem JAR-File mit dem Namen `erv_sta_validation_<Versionsnummer>.jar` gebündelt werden.

3.2 Basic Flow

Nach Aufruf der Validierung gemäß Kap. 3.3.1 mit den gemäß Kap. 3.3.2 und 3.3.3 befüllten Objekten ergibt sich folgender Ablauf im Validierungsmodul:

1. Prüfung des Inputs auf Existenz der Minimalanforderungen gemäß Kap. 3.3.5. Bei Fehlern wird die Validierung ohne Prüfung durch Geschäftsregeln abgebrochen.
2. Prüfung, ob Input „well-formed“ ist. Ist dies nicht der Fall, stellt dies eine Verletzung der Geschäftsregeln BR-0008 bzw. BR-0032 dar. Die Prüfung der anderen Geschäftsregeln entfällt.
3. Validierung des Inputs gemäß den veröffentlichten Geschäftsregeln.
4. Als Ergebnis wird im Fehlerfall eine Liste der Fehlermeldungen gemäß Kap. 3.3.6 zurückgegeben.

3.3 Application Programming Interface (API)

Im folgenden Kapitel werden die öffentlichen APIs des Validierungsmoduls näher beschrieben. Damit soll es möglich sein, die Validierung einer Nachricht aufrufen und das zurückgegebene Ergebnis interpretieren zu können.

3.3.1 Class StaEingabeValidierung

Die Validierungsklasse *`ErsteValidierung`* enthält die Prüfmethode *`validate()`*, welche den applikationsspezifischen Inhalt der eingehenden Nachricht (gekapselt in einer *`Message`*) prüft.

```
package at.gv.brz.bjujq.sta.validator.uest
public final class ErsteValidierung {
   public static ValidationResult validate(final Message msg)
     throws ValidationException
   public static Version getVersion();
}
```

Die Prüfmethode *`validate()`* ist synchron und liefert normalerweise ein *`ValidationResult`* (welches eine Liste von Validierungsergebnissen des Typs *`Result`* enthält) zurück. Im Ausnahmefall wird eine *`ValidationException`* geworfen.

Die Nachricht (*msg*) gilt nur dann als valide, wenn `ValidationResult.isOk()` dies bestätigt.

Die Methode *getVersion()* gibt in einem Objekt der Klasse *Version* die strukturierte Versionsinformation für das Validierungsmodul zurück.

3.3.2 Interface Message

Für eine übermittelte Nachricht muss ein Objekt erzeugt werden, welches das Interface *Message* implementiert.

```
package at.gv.justiz.erv.commons.validation.api;
interface Message {
   List /*<Part>*/ getParts();
   Date getEinbringungszeitpunkt();
   Date getErstellzeitpunkt();
   String getMessageId();
   String getAnwendungsKennung();
   String getAnschriftcode();
}
```

Eine *Message* besteht aus einem oder mehreren *Parts*. Außerdem sieht es eine Reihe[1] von `String` und `Date` Properties vor, dies sich von der SOAP Nachricht ableiten.

3.3.3 Interface Part

Jeder in einer Nachricht übermittelte Anhang wird zu einem *Part*, dh. die Payload bzw. alle anderen Anhänge werden als Parts abgebildet.

```
package at.gv.justiz.erv.commons.validation.api;
interface Part {
   InputStream getContent();
   PartArt getArt();
   String getAnhangId();
   String getReferenzId();
   String getContentType();
}
```

So wie *Message*, bietet *Part* eine Menge von simplen Properties an. Ein *Part* bietet auch über *getContent()* die Möglichkeit, an seinen Inhalt zu kommen.

3.3.4 Mapping Mime-Felder / Message/Part-Felder

Nachstehend befindet sich eine Zuordnungstabelle von Mime Feldern zu Message/Part Feldern. Auch kann diese Zuordnung aus den korrespondierenden Interfaces mittels Javadoc ausgelesen werden.

[1] Siehe JavaDoc bzw. Tabelle in Kap. 3.3.4

Mime Header	Mime Part	Message	Part	Pflicht
Teilnehmer		getAnschriftcode()		x
Message-ID		getMessageId()		x
Anwendungskennung		getAnwendungskennung()		x
Einbringungszeitpunkt		getEinbringungszeitpunkt()		
	Content-ID		getAnhangId()	x
	Referenz-ID		getReferenzId()	
	Art		getArt()	x
	Xml Content		getContent()	x

Tabelle 1: Zuordnung Mime-Felder / Message/Part-Felder

3.3.5 Minimalanforderungen

Die Methode *`validate()`* liefert „INIT_0001=Invalid Input error {0}“ zurück, wenn nicht alle der folgenden Daten vorhanden sind:

1. `Message`
2. `Message.getAnwendungskennung()`
3. `Message.getAnschriftCode()`
4. `Message.getMessageID()`
5. `Message.getMessageParts()`
6. `MessagePart.getArt()`
7. `MessagePart.getData()`
8. `MessagePart.getAnhangID()`

wobei für die *Anwendungskennung* gilt, dass der Wert „STA“ gesetzt sein muss.

3.3.6 Interface Result

Ein *`Result`* wird zurückgegeben, um ein definiertes Ergebnis der Validierung einer Nachricht zu beschreiben.

```
package at.gv.justiz.erv.commons.validation.api;
interface Result {
   String getPublishedId();
   ResultKind getKind();
   String getMessage();
}
```

Der Wert von *`getPublishedId()`* stellt die Kennung der jeweiligen Fehlermeldung dar.

Die Methode *`getKind()`* beschreibt mit einem *`ResultKind`* die Art des Resultats. *`ResultKind`* ist ein „Type-Safe Enumeration“. **Bei einem *`Result`* mit *`getKind()`* == *`ResultKind.ERROR`* gilt die Message als invalid.**

Die Methode *`getMessage()`* liefert eine zur Anzeige geeignete Fehlermeldung. Die Fehlermeldungen sind in ***STA_Fehlermeldungen*** beschrieben und werden durch ihre Kennung (vgl. *`getPublishedId()`*) identifiziert.

3.3.7 Interface ValidationResult

Für die STA-Validierung sind folgende Listen implementiert:

- getSatisfiedRules() – Liefert ein Set mit den erfüllten Regeln.
- getViolatedRules() – Liefert ein Set mit den verletzten Regeln.
- getResults() – Liefert alle Resultate (Satisfied und Violated)

3.3.8 Class Version

Die *Version* enthält die strukturierte Information, welche Version das Validierungsmodul hat.

```
package at.gv.justiz.erv.commons.validation.api;
public class Version implements Comparable, Serializable {
   ...
   public int compareTo(Object that);
   pubilc String toString();
}
```

Versionen lassen sich sortieren und vergleichen. Über *toString()* bekommt man eine kurze Identifikation der jeweiligen Validierung.

3.4 Mitgelieferte Komponenten

Die Validierung selbst wird als jar Datei ausgeliefert:

- erv_sta_validation_<Versionsnummer>.jar.

Die passenden Teile unseres „ERV Commons“ Frameworks werden ebenfalls mitgeliefert:

- erv_commons-<Versionsnummer>.jar

Der Name kann je nach Version und Buildnummer variieren.

3.5 Externe Abhängigkeiten

Das Validierungsmodul verwendet Java in der Version 1.6.x. Darüber hinaus werden einige andere Open Source Bibliotheken verwendet, die in folgender Tabelle aufgelistet sind.

Bibliothek	Version	Jar	Quelle bzw. Kommentar
Xerces	2.9.1	xml-apis.jar xercesImpl.jar	http://xerces.apache.org/xerces-j
jUnit	4.8.1	junit.jar	http://junit.sourceforge.net/junit3.8.2/index.html (Wird benötigt um den Selbsttest auszuführen.)
Commons-lang	2.6	Commons-lang	https://commons.apache.org/proper/commons-lang/
Slf4j	1.6.4	Slf4j-api.jar slf4j-simple.jar	http://www.slf4j.org/

Tabelle 2: Verwendete externe Bibliotheken

Alle in obiger Tabelle erwähnten Bibliotheken müssen für eine einwandfreie Funktion des Validierungsmoduls im CLASSPATH eingetragen sein.

3.6 Selbsttest

Als Hilfsmittel um die korrekte Einrichtung zu verifizieren, steht eine Selbsttestfunktion zur Verfügung. Diese macht nichts anderes als einen Unittest der Eingabevalidierung durchzuführen. Sollte dieses ohne Fehler gelingen, ist die Einrichtung (CLASSPATH usw.) korrekt.

```
package at.gv.brz.bjujq.sta.validator.uest;

public final class SelfTest {

    public static void selfTest(); { … }

    public static void main(String[] args) {

       junit.textui.TestRunner.run(SelfTestUnitTest.class); }

     }
```

Ein Aufruf von der *Windows* Commandline sieht so aus:

```
# wir stehen in einem Verzeichnis mit den benötigten jars

$ java -cp xercesImpl-2.9.1.jar;junit-4.8.1.jar;slf4j-api-
1.6.4.jar;slf4j-simple-1.6.4.jar;commons-lang-2.6.jar;erv_commons-
1.3.2.jar;erv_sta_validation_1.0.12.jar
at.gv.brz.bjujq.sta.validator.uest.SelfTest


Time: 0,017

OK (1 test)
```