7437 - EDI und E-Business Standards

Praktikumsaufgabe 06:

Bestellungen und Lieferavise mit UN/EDIFACT und EANCOM,

Mapping per EDI-Bibliothek

Das Szenario

- Szenario
  - Die Praktikumsteilnehmer setzen ihre Handelspartner-Rolle (Lieferanten, Händlern) der Konsumgüterbranche aus Praktikum 01/02 fort.
  - Sie korrigieren ggf. die in Übung 05 entworfenen Beispieldaten für Bestellungen und Lieferavise und entwickeln ein Programm zur automatischen Erzeugung dieser Daten.
- Ziele der Übung
  - Praxis im Umgang mit einer Klassenbibliothek für EDI-Daten
  - Beherrschung der EDIFACT Service-Segmente
  - Mappingtechnik-Grundlagen

Mapping

- Die Aufgabe
  - Mappen Sie Ihre ausgehenden Belege aus Übung 04 (Händler: Bestellungen, Lieferanten: Lieferavise)
    - von Ihrer In-house Struktur aus Übung 03/04
    - nach EANCOM '02 gemäß des (korrigierten) Nachrichtenaufbaus aus Übung 05
  - Programm-Name:

    `mapper06.rb`
  - Gewünschter Aufruf:

    `mapper06.rb` *`srcdata.msg`* `>` *`eancomdata.edi`*

- Ausblick
  - Übung 07 behandelt die eingehenden Nachrichten.

Mapping

- Ersatz-Aufgabe
  - Wer die Aufgaben bisher nicht (hinreichend) bearbeitet hat, kann dennoch hier "einsteigen"!
  - **Mappen Sie ggf. (statt Ihrer eigenen Daten) Beispiel-Bestelldaten im Format von Metro MGI's "MEC" WebEDI-Lösung nach EANCOM '02**

- Einzelheiten
  - Formatbeschreibung Quellformat:
    - ~werntges/lv/edi/ss2003/proj/MEC-Schnittstelle-ORDERS.pdf
  - Beispiel-Quelldaten:
    - ~werntges/lv/edi/ss2003/proj/bestell-khof1.mec
    - Daten bitte an Ihr Szenario anpassen (GLNs, GTINs!)

Mapping

- **Hinweise zum Ablauf**
  - Erzeugen Sie zunächst das noch leere Interchange
    - Sender- und Empfänger-GLN können mit denen aus den NAD-Segmenten übereinstimmen.
    - Aktuelles Datum befüllen
    - UNB Interchange Reference mittels Nummerngenerator aus Übung 01 befüllen.
  - Legen Sie dann eine (leere) EDIFACT-Nachricht an
    - Header-Angaben fest vorgeben, eingebauten Nachrichtenzähler gewähren lassen (d.h. Sie greifen möglichst nicht in UNH und UNT ein).
  - Befüllen Sie die Nachricht nun segmentweise
    - Achten Sie dabei auf die korrekte Reihenfolge der Segmente.
      Ggf. Segmente vor dem add()-Aufruf in einem Array zwischenlagern.

Technische Hinweise

- Entwickeln mit Ruby: Allgemeines
  - Erstellen Sie Dateien "*.rb"
    - Editoren: emacs (besitzt Ruby-Modus), SciTE (scite), ...
    - Zum Testen: irb (Interactive Ruby), eine sehr praktische Ruby-Shell
  - Ausführung auf der Kommandozeile, etwa:

```
$ ruby sample.rb
```

Erste Zeile = `#! /usr/bin/env ruby` , dann: `chmod +x sample.rb`

```
$ ./sample.rb          # nun direkt ausführbar.
```

Technische Hinweise

- Dateikopf (Einbindung der Bibliothek):

```
#!/usr/bin/env ruby
require "rubygems"
require_gem "edi4r"
```

- Erzeugung eines EDIFACT Interchanges:

```
ic = EDI::Interchange_E.new({
          'charset' => 'UNOA',
          'version' => '2',
          'intref'  => '123456',
          'output_mode' => 'formatted' # oder ''
     })
cde = ic.header.cS002              # S002 aus UNB
cde.d0004 = "1234567890123"        # GLN zuweisen
cde.d0007 = "14"                   # Qualifier setzen
# usw. für alle relevanten Header-Daten
```


Technische Hinweise

- Erzeugen und Einfügen einer Nachricht (Bsp. ORDERS):

```
params = {
  'msg_type' => 'ORDERS',
  'version' => 'D',
  'release' => '01B',
  'resp_agency' => 'UN',
  'assigned_code' => 'EAN010'
}
msg = ic.new_message( params )
# Befüllen mit Segmenten, ..., schließlich:
ic.add msg      # Nachricht an Interchange anfügen
```

Technische Hinweise

- Segmente erzeugen und befüllen:

```
seg = msg.new_segment( "BGM" )
seg.cC002.d1001 = '220'
seg.cC106.d1004 = '123456abc'     # Belegnummer
seg.d1225 = '9'                   # Qualifier setzen
# usw., schließlich:
msg.add seg     # Segment an Nachricht anfügen
```

- Kopfdaten beeinflussen:

```
msg.header.d0062 = "543210" # UNH, DE 0062 setzen

# msg.trailer existiert zwar, sollte aber automatisch
# korrekt befüllt werden - nicht beschreiben!
```


Technische Hinweise

- Mehrfach vorkommende Datenelemente adressieren:
  - Kommen Composites und einfache Datenelemente mehrfach vor, bilden sie einfach ein Array (Achtung – Index beginnt bei 0):

```
seg = msg.new_segment( "PIA" )
cde = seg.cC212[0]    # seg.cC212 ist hier ein Array von CDEs
cde.d7143 = 'SA'      # SA = supplier assigned
```

  - Ein Element aus einem Array einfacher Datenelemente ist nicht – wie sonst – mit seinem Inhalt gleichgesetzt. Auf Inhalte eines DE-Objekts greift man mit der Methode „value“ zu:

```
seg = msg.new_segment( "NAD" )
seg.cC080.d3036[0].value = 'MeineFirma GmbH'
```

Technische Hinweise

- Erzeugte Daten ausgeben:

```
ic.validate    # Gegen Directories prüfen lassen
ic.write $stdout # Auch Teilausgaben: msg.write hnd

# Schreibt den Interchange nach STDOUT
# Wenn output_mode == 'formatted':
#        Eine Zeile pro Segment
# Sonst: Als character stream, wie im Standard.

# In Datei schreiben:
#   Datei vor Aufruf öffnen, handle übergeben:
ic.output_mode = ''
File.new("my_output.edi","w") {|hnd| ic.write hnd }
```


Abgaben

- Abzugeben
  - `mapper06.rb` # Ihr Programm-Code
  - `srcdata06.msg` # Ihre verwendeten Quelldaten
  - `eancomdata06.edi` # Ihre Ergebnisse

  (srcdata06.msg und eancomdata06.edi sollten von Ihrem Programm verwendet bzw. erzeugt worden sein. Sie sollten Ihren Dateiformaten aus Übung 03 und 05 entsprechen.)

- Abgabeordner
  - Wie üblich, unter `~werntges/lv/edi/abgaben/a/`*`<matrnr>`*

- Annahmeschluss
  - Kein formaler Annahmeschluss (außer im Rahmen der Projektabnahme), aber Programm wird von der übernächsten Übung benötigt!