



7437 - EDI und E-Business Standards

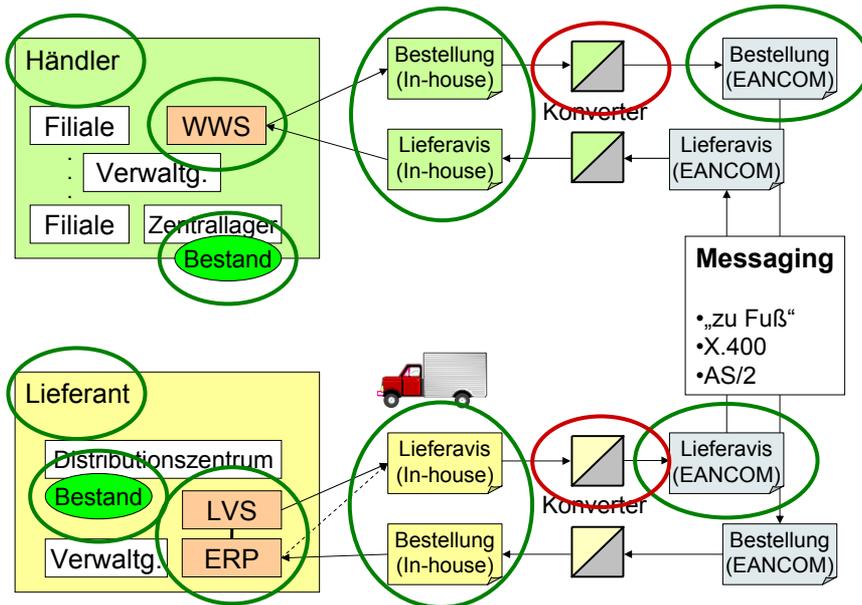
Praktikumsaufgabe 06:
Bestellungen und Lieferavis mit
UN/EDIFACT und EANCOM,
Mapping per EDI-Bibliothek



Das Szenario



- Szenario
 - Die Praktikumssteilnehmer 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 Lieferavis 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



- 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.
 - Verwenden Sie Ihre EANCOM-CDROM zur Klärung aller Details!
 - Programm-Name:
mapper06.rb
 - Gewünschter Aufruf:
mapper06.rb srcdata.msg > eancomdata.edi
- Ausblick
 - Übung 07 behandelt die eingehenden Nachrichten.



- **Hinweise zum Ablauf**
 - Erzeugen Sie zunächst das noch leere *Interchange*
 - Sender- und Empfänger-GLN des UNB-Segments können mit denen aus den NAD-Segmenten übereinstimmen.
 - Aktuelles Datum befüllen
 - *UNB Interchange Control 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.



- **Entwickeln mit Ruby: Allgemeines**
 - Erstellen Sie Dateien "*.rb"
 - Editoren: emacs (besitzt Ruby-Modus), SciTE (scite), Kate, ...
 - 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.
```



- Dateikopf (Einbindung der Bibliothek):

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

- Erzeugung eines EDIFACT Interchanges:

```
ic = EDI::E::Interchange.new({
  :charset => 'UNOA',
  :version => 2,
  :interchange_control_reference => '123456',
  :output_mode => :linebreak # oder :verbatim
})

cde = ic.header.cs002          # S002 aus UNB
cde.d0004 = "1234567890123"  # GLN zuweisen
cde.d0007 = "14"             # Qualifier setzen
# usw. für alle relevanten Header-Daten
```



- 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
```



- 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!
```



- 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.aC212[0]      # seg.aC212 ist 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.a3036[0].value = 'MeineFirma GmbH'
```



- Erzeugte Daten ausgeben:

```
ic.validate      # Gegen Directories prüfen lassen
print ic        # Auch Teilausgaben: print msg

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

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



- Weitere Hinweise:

- Datei „minidemo.rb“ im Ordner ~werntges/lv/edi/06
- Vorlesungsbeispiele im Kapitel „EDIFACT“
- Tutorial im Gem „edi4r-0.9.3“ (Demo)
 - \$ gem_server # GEM Server starten
 - Mit Browser den angegebenen Port auf „localhost“ besuchen (8808?)
 - Gem auswählen, nachlesen
- Weitere Einzelheiten aus der RDoc-Doku des Gems entnehmen

- Support anderer UN/TDIDs

```
require_gem "edi4r-tdid"
```



- **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**
 - Freitag vor der nächsten Übung (d.h. nach Pfingsten).
- **Hinweis:**
 - Programm wird von der übernächsten Übung benötigt!