7437 - EDI und E-Business Standards

Electronic
Data
Interchange
(Elektronischer Datenaustausch)

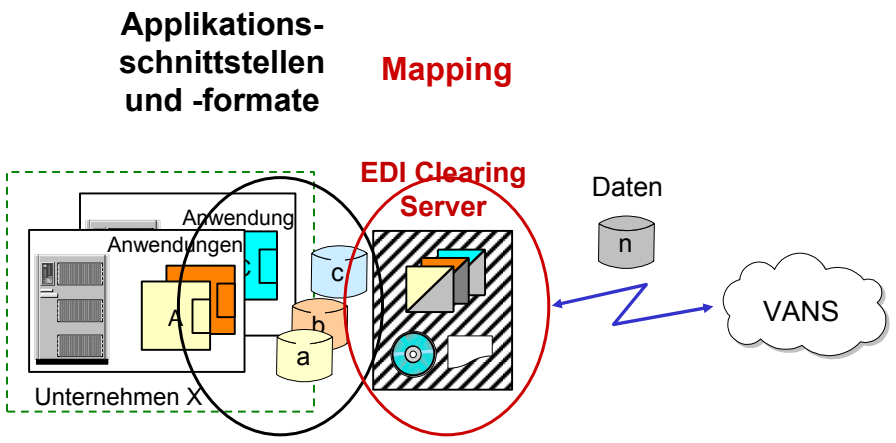
Klassisches EDI - der Kern

Einleitung - die Kernkomponenten
File Transfer- und Messaging-Standards
UN/EDIFACT und EANCOM im Detail
Applikationsschnittstellen
Konverter- und Mappingtechniken

EDI: Die Kernkomponenten

- ✓ EDI-Standardaustauschformate
- **Applikationsschnittstellen**
- **Mapping**
- ✓ Routing
- ✓ Messaging / File Transfer
- Extras
  - Archivierung
  - Reporting
  - Alarmierung
  - Tracking & Tracing

Einordnung



Applikationsschnittstellen

Schnittstellenarten

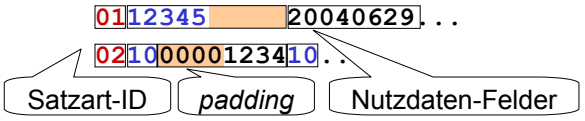
Die SAP IDoc-Schnittstelle - ein "prominentes" Beispiel

Methoden der Inhouse-Anbindung

- **Dateischnittstellen**
  - die traditionelle Methode
- Austausch von Speicherstrukturen durch **IPC-Verfahren**
  - für zeitkritische Anforderungen
- Direkte Kopplung über **Datenbankzugriffe** auf Applikationstabellen
  - nur in speziellen Umgebungen geeignet
  - risikoreich aus Applikationssicht
  - nicht gerade modular
  - elegant, wo der Einsatz vertretbar ist

Dateischnittstellen: Datenformat-Arten

- ***Fixed record*-Strukturen**
  - Positiv:
    - einfach
    - schnell in der Verarbeitung
    - gut zu parsen
  - Negativ:
    - unflexibel bei späteren Anwendungen
    - verschwendet viel Platz (*padding* mit Blanks oder Nullen)
  - Beispiele:



    - In der Praxis eingesetzt von: SAP IDocs; SEDAS, GENCOD

Dateischnittstellen: Datenformat-Arten

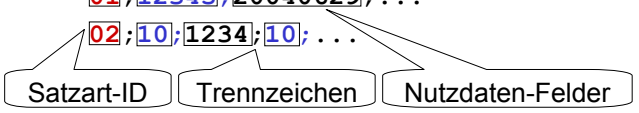
- ***Fixed record*-Strukturen: Lesetechnik. Ein einfaches Ruby-Beispiel:**

```
class MyRecs
  @@fldLen =  {'01'=>[10,8],           '02'=>[2,8,2]}
  @@fldType = {'01'=>['an..10','an8'], '02'=>['n2','n..8','n..2']}

  def MyRecs.split( rec )
    rid = rec[0..1]; a = [ rid ]; offset = rid.length
    @@fldLen[rid].each_with_index do |len, i|
      next_offset = offset + len
      field = rec[offset...next_offset]
      case @@fldType[rid][i]
        when /^n.*/;        a << field.to_i
        when /^an\d+/;      a << field
        when /^an\.\.\d+/;  a << field.sub(/\s$/,'')
      end
      offset = next_offset
    end
    a
  end
end
p MyRecs.split "0112345     20040629"     # Tests
P MyRecs.split "02100000123410"
```

Dateischnittstellen: Datenformat-Arten

- *Variable record*-Strukturen
  - Positiv:
    - Kompakt
    - Flexibel bei Feldlängenänderungen
  - Negativ:
    - Trennzeichen erforderlich (verkomplizieren das Parsen: ESC-Mechanismus erforderlich (ESC = "*escape*"))
    - Nur seriell verarbeitbar (!)
  - Beispiel:
    - CSV (*comma separated variables*), TSV (*tab separated v.*):



Dateischnittstellen: Datenformat-Arten

- ***Variable length record*-Strukturen: Lesetechnik. Ein einfaches Ruby-Beispiel:**

```
# Für CSV muss man praktisch nicht programmieren:

# Mit Ausgabe der Satzart-ID:
p "01;12345;20040629".split(';')

# Ohne Ausgabe der Satzart-ID:
p "02;10;1234;10".split(';').slice(1..-1)
```

Dateischnittstellen: Datenformat-Arten

- Sonstige: *key/value*-Listen
  - Auch "*stanzas*" bzw. Strukturen wie Windows *.ini-Dateien
  - Positiv:
    - sehr flexibel, auch bei Aufnahme neuer Felder
    - selbst-dokumentierend (über sprechende *keys*)
  - Negativ:
    - Overhead
    - kein *a priori*-Wissen über den Ort eines erwarteten Wertes
  - Beispiel:

```
recID=header,orderNo=12345,orderDate=20040629, …
recID=item,matNo=1234,quantity=10,...
```


Dateischnittstellen: Datenformat-Arten

- ***Key/value*-Listen: Lesetechnik. Ein einfaches Ruby-Beispiel:**

```
class MyRecs
  @@fieldSep, @@kvSep = ',', '='  # i.A. noch "escaping" notwendig

  def MyRecs.to_h( rec )  # Record in Hash umwandeln
    h = Hash.new
    rec.split(@fieldSep).each do |kv|
      key, value = kv.split(@@kvSep)
      h[key.strip] = value.strip
    end
    h
  end
end

# Tests:
p r01=MyRecs.to_h("recID=header,orderNo=12345,orderDate=20040629")
p r02=MyRecs.to_h("recID=item, matNo = 1234 , quantity= 10")
```

Dateischnittstellen: Datenformat-Arten

- ***Key/value*-Listen: Codelisten, Umgang mit Teilfeldern**
  Ein einfaches Ruby-Beispiel (Forts.):

```
Month = {                  # Ein Hash als Codeliste
   '01'=>'Jan', '02'=>'Feb', '03'=>'Mar', '04'=>'Apr',
   '05'=>'May', '06'=>'Jun', '07'=>'Jul', '08'=>'Aug',
   '09'=>'Sep', '10'=>'Oct', '11'=>'Nov', '12'=>'Dec'
}

# Extraktion von Teilfeldern, Mapping per Codeliste 'Month':

re = Regexp.new('(\d{4})(\d{2})(\d{2})') # YYYYMMDD erwartet
md =       re.match(r01['orderDate'])    # Ein MatchData-Objekt…
printf "Oder date is: %s-%s-%s\n", md[3], Month[md[2]], md[1]
```

- **Merke:**
  - Codelisten = Tabellen, die i.a. Codes auf Bedeutungen abbilden
  - Sie lassen sich oft direkt als Hashes implementieren!

Dateischnittstellen: Datenformat-Arten

- Sonstige: *Markup*, insb. XML
  - Positiv:
    - ideal für hierarchische *record*-Strukturen
    - selbst-dokumentierend
      - in einfacher Form über sprechende *tags*
      - ggf. auch detailliert, über DTD bzw. Schema
    - validierbar
    - Maßgeschneiderte Datentypen möglich (per XML Schema)
  - Negativ:
    - sehr großer *overhead*
    - hoher Speicherverbrauch - massendatentauglich?
    - komplexes Interface (DOM)
    - Eigene Selektionssprachen (XPath, XQuery)

Dateischnittstellen: Datenformat-Arten

- Beispiel zu XML (fiktiv)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<DOCTYPE my_orders ...>
<my_orders>
  <header>
    <order>
      <refno>12345</refno>
      <date fmt="...">20040629</date>
    </order> <!-- ... -->
  </header>
  <items>
    <item no="10">
      <matNo>1234</matNo>
      <quantity unit="pieces">10</quantity>
    </item> <!-- etc. -->
  </items>
<my_orders>
```


Dateischnittstellen: Datenformat-Arten

- ***XML-Dokumente*: Lesetechnik. Ein einfaches Ruby-Beispiel:**

```
require 'rexml/document'

xmldoc = REXML::Document.new(File.open("myorders.xml")

# Zugriff auf XML-Element z.B. mit XPath:
orderDate = xmldoc.elements["/my_orders/header/order/date[1]"].text

# Ausgabe analog zu key/value-Beispiel:
mtch = Regexp.new('(\d{4})(\d{2})(\d{2})').match(orderDate)
puts "Oder date is: %s-%s-%s" % [mtch[3], Month[mtch[2]], mtch[1]]

# Noch ein XML-Zugriff per XPath:
matNo = xmldoc.elements["/my_orders/items/item[@no='10']/matNo"].text
puts 'Mat no. of item 10 is "%s"' % matNo
```

Dateischnittstellen: Datenformat-Arten

- Bemerkungen zum Begriff ***flat file***
  - Vorsicht - keine einheitliche Verwendung!
  - Zwei recht konträre, aber gebräuchliche Bedeutungen:

    a) serialisierte Speicherstruktur eines komplexen Typs
    (etwa: "flachgeklopfte Hierarchie")

    b) 1:1-Abbildung des EDIFACT *interchange*
    als *fixed record*-Format, 1 *record* pro Segment

Die SAP EDI-Schnittstelle- ein "prominentes" Beispiel

Das IDoc-Konzept
IDoc-Struktur
IDoc-Verwaltung
Besonderheiten, Ausblick

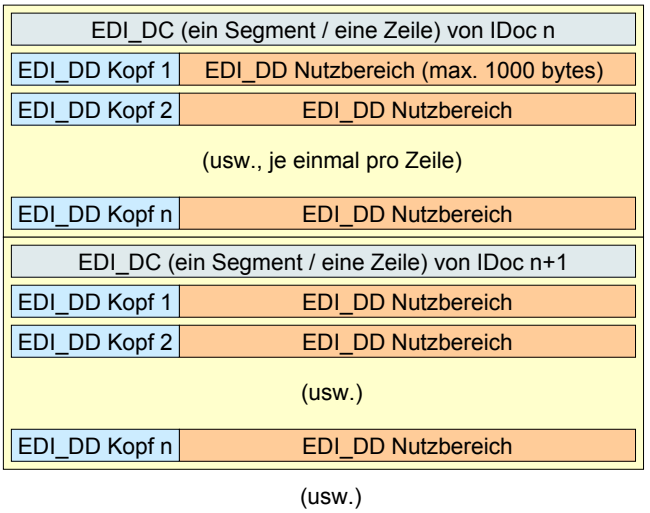
Das IDoc-Konzept

- IDoc = *intermediate document* = Zwischenbeleg
- Eine Struktur...
  - zwischen Anwendungsbeleg einerseits
  - und klassischem, sehr dem EDI-Format nachempfundenen "*flat file*“
- Kernidee:
  - Abstraktionsschicht oberhalb konkreter EDI-Standards wie UN/EDIFACT oder ANSI X12
- Einheitliche Kommunikationsform zwischen
  - verschiedenen SAP-Systemen bzw.
  - SAP und sog. “Sub-Systemen”, insb. EDI-Servern

IDoc-Struktur

- **EDI_DC**
  - Allgemeiner Verwaltungskopfsatz
  - In etwa analog zu einer Kombination aus UNH und UNB
  - Geht jedem IDoc genau einmal voraus
- **EDI_DD**
  - Verwaltungsrahmen für Nutzsegmente
  - Nutzsegmente (je max. 1000 Bytes):
    - mit Versionsnummer
    - fester Satzaufbau
    - Hierarchie
    - Erweiterbarkeit
- **EDI_DS**
  - Statussatz, für Rückmeldungen ausgehender Belege

Struktur einer IDoc-Datei, schematisch



IDoc-Struktur

- XML-IDocs
  - Gleicher Inhalt, neue Verpackung
  - Eher noch größeres Datenvolumen als bei den bereits recht großen IDocs mit *fixed record*-Struktur
- **IDoc-Dokumentation** - *online*-Beispiele
  - HTML-Dokumentation, wie man sie aus SAP R/3 exportieren kann
  - a) EDI_DC, EDI_DD, EDI_DS
  - b) Nutzdaten-Beispiel: ZINVOIC1

IDoc-Verwaltung

- Übergabetabellen (DB)
- Eigene Statusverwaltung der IDocs
  - intern wie extern
  - mit eigenem *user interface*
- Dokumentation in IDoc-Verwaltung integriert
  - IDoc-Formatbeschreibungen sind jederzeit exportierbar
    - in strukturierter Form, vergleichbar XML-DTD/Schema
    - als HTML-Dokumentation
    - als „C"-Headerdateien (Makros, *struct*'s, …)
- Technische Schnittstellen
  - EDI (i.w. Dateischnittstelle plus *RFC* (SAP's RPC-Methode)
  - ALE („*application link enabling*") - *high-level* RFC

Die technische EDI-Schnittstelle

- EDI Ports - oder: SAP erreicht das Betriebssystem
  - In SAP werden (logische) EDI-Ports angelegt
  - Auf Betriebssystemseite werden ihnen Unterverzeichnisse zugewiesen
- Typische derartige Verzeichnisstruktur:
  - Basis-Verzeichnis
    - z.B. auf einem Unix-Host: **`/usr/sap/edi/<SAPSYS>/<PORTNAME>`**
    - Konkret etwa (fiktiv): **`/usr/sap/edi/PW1/FB06`**

      Produktionsystem, FH **W**iesbaden, Server **1**

      Port des Fach**b**ereichs **06**
  - Unterverzeichnisse:

    `./in` für eintreffende IDoc-Dateien

    `./out` für ausgehende IDoc-Dateien

    `./status` für eingehende Dateien mit Statussätzen

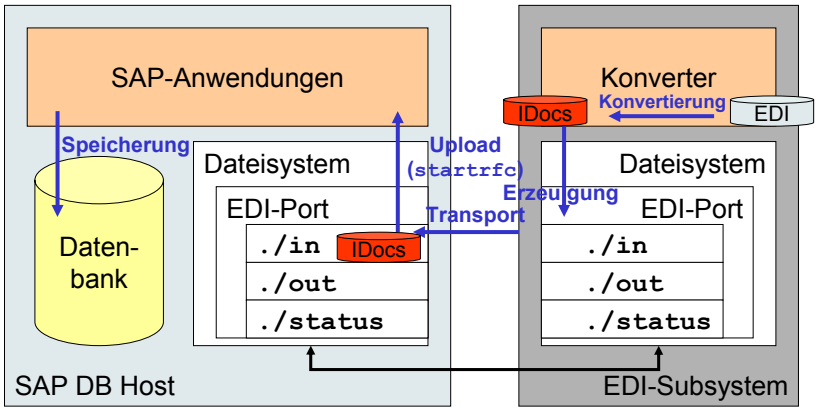
Die technische EDI-Schnittstelle

- Signalisierung bei ausgehenden Daten:
  - Drei Betriebsarten:
    - Polling – keine Signalisierung
      - SAP erzeugt IDoc-Dateien in ./out
      - EDI-Subsystem holt sie zeitgesteuert ab
      - Einfach, aber mit Kollisionsgefahr. Für Batchbetrieb ausreichend
    - Triggern pro Exportdatei
      - SAP erzeugt IDoc-Dateien in ./out
      - SAP startet Folgeverarbeitung für jede Datei mittels rfcexec-Aufruf
      - Effizient, guter Kompromiss, Regelfall. Komplexere Systemkopplung
    - Triggern für jedes IDoc
      - SAP erzeugt für jedes IDoc eine eigene Datei in ./out
      - SAP startet Folgeverarbeitung für jedes IDoc mittels rfcexec-Aufruf
      - Hohe Systemlast, geeignet für sehr zeitkritische Abläufe, ALE-artig.
  - **rfcexec:** Nahtstelle zwischen SAP-Programmen und dem BS
    - rfcexec ist ein von SAP bereitgestelltes, auf dem jeweiligen BS lauffähiges Binärprogramm, das aus SAP heraus per RFC gestartet wird.
    - Seine Aufgabe: Ausführung „beliebiger“ externer Programme / **Scripte**

Die technische EDI-Schnittstelle

- Signalisierung bei eingehenden Daten:
  - Zwei Betriebsarten:
    - Polling – keine Signalisierung
      - EDI-Subsystem erzeugt IDoc-Dateien in ./in bzw. Statussatz-Dat. in ./status
      - SAP holt sie zeitgesteuert ab
      - Einfach, aber mit Kollisionsgefahr. Für Batchbetrieb ausreichend
    - Triggern pro Eingangsdatei
      - EDI-Subsystem erzeugt IDoc-Dateien in ./in bzw. Statussatz-Dat. in ./status
      - … und startet Folgeverarbeitung in SAP mittels startrfc-Aufruf
      - Der Regelfall. Konfiguration der zahlreichen startrfc-Optionen nicht immer einfach.
  - **startrfc:** Nahtstelle zwischen BS und SAP-Programmen
    - startrfc ist ein von SAP bereitgestelltes, auf dem jeweiligen BS lauffähiges Binärprogramm.
    - Es wird von externen Programmen (typisch sind Scripte) aufgerufen, um in SAP per RFC bestimmte Funktionsmodule aufzurufen, etwa die zum IDoc-Import und deren Verarbeitung.

Die technische EDI-Schnittstelle: Eingehende Daten



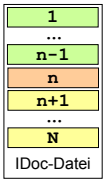
Kopplung, z.B. per NFS

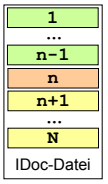
Die technische EDI-Schnittstelle

- Fehlerbehandlung
  - Ausgehende Daten:
    - Verfolgung von Fehlern auf IDoc-Ebene durch Abfolge von Statuswerten
      - Interne Statuswechsel protokolliert SAP automatisch
      - Externe Statuswechsel muss das EDI-Subsystem per Statussatz mitteilen
      - Beispiele: Subsystem getriggert, IDoc übersetzt / übermittelt, *func. ackn.*
    - *WICHTIG: SAP-interne Überwachung der Ende- und Fehler-Zustände*
  - Eingehende Daten
    - **Grundregel**:
      Datei von SAP gelöscht = Verantwortung an SAP übergeben
    - Rückgabewerte von **`startrfc`** nicht erhältlich bzw. nicht aussagekräftig!

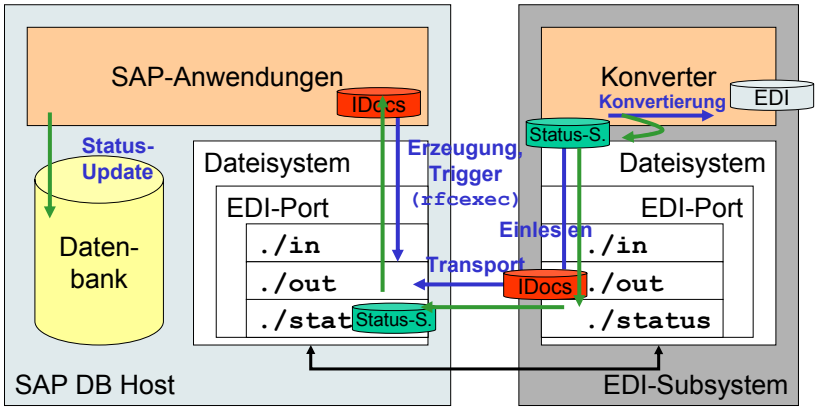
Die technische EDI-Schnittstelle

- Problemfall bei eingehenden Daten:
  - Datei enthält mehrere IDocs, Übernahme scheitert
  - Problem:
    - Welche IDocs wurden bereits von SAP übernommen (und dürfen daher nicht erneut eingespielt werden!) ?
    - Welche müssen repariert bzw. ausgesondert werden?
    - Mit welchen IDocs darf (bzw. muss schleunigst) ein erneuter Versuch starten?
  - Lösung 1 (manuell und mühsam!)
    - Manuell in SAP nachsehen, für welche IDocs die Übernahme gelang
    - Diese (die ersten *n-1*) aus der IDoc-Datei entfernen
    - IDoc *n* ist vermutlich defekt – ebenfalls aussondern, Fehler analysieren
    - Erneuter Übernahmeversuch mit dem Rest (*n+1 … N*)
  - Lösung 2 (in der Praxis bewährt)
    - Eingehende IDoc-Dateien splitten:
      - Ein IDoc pro Datei
      - SAP für jedes IDoc separat triggern
      - Aussondern der Fehlerfälle + Alarmierung ist dann einfach
    - Nachteil: Höhere Systemlast seitens SAP
      - Beherrschbar, solange nur IDoc-Übernahme und nicht auch Verbuchen getriggert wird.





Die technische EDI-Schnittstelle: Ausgehende Daten



Kopplung, z.B. per NFS

Die technische EDI-Schnittstelle

- ALE Ports
  - Austausch von Speicherstrukturen direkt über ein Netzwerkprotokoll, ohne Dateisystemkontakt
  - Erfordert ALE-Fähigkeit auf beiden Seiten
  - Gut insbesondere für zeitkritische Anwendungen
  - ALE wird insbesondere zum Belegaustausch zwischen SAP-Systemen verwendet, seltener zwischen SAP und Subsystemen wie EDI-Servern.
  - Strukturanpassungen notwendig, falls Kopplung zwischen SAP-Systemen mit unterschiedlichen *release*s,
    - etwa R/3 4.6 zu koppeln mit R/3 3.1

Die technische EDI-Schnittstelle

- Besonderheiten
  - Bildung von Applikationsserver-Gruppen
    - **`startrfc`** wendet sich an einen *messaging server* (am besten auf dem ohnehin stets verfügbaren DB-Server betrieben),
    - dieser bestimmt Servermitglied aus "EDI-Gruppe" für den Import
  - Vorteile
    - Lastverteilung
    - Risikostreuung
    - Permanente Verfügbarkeit der EDI-Schnittstelle auch wenn ein Applikationsserver gewartet wird

Die technische EDI-Schnittstelle

- Ausblick / Verwandte Themen
  - BAPIs (Business APIs)
    - genormte *high-level* Schnittstellen z.B. zur Erzeugung eines kompletten Geschäftsdokuments durch ein externes System
  - Business Integration Server (MOM, XML, *Web services*)
  - SAP Business Connector
    - Kostenlose Zusatzkomponente, Einstiegshilfe in XML-Tech / WS
    - Reduzierte Version von einem Produkt von WebMethods
  - SAP NetWeaver
    - SAP's neue eigene Integrations-Technologie
    - SOAP-basierte Web Services bilden eine wesentliche Grundlage