Inhaltsverzeichnis

1. Vorwort

Dieses Projekt ist total spannend für mich, ich freue mich schon jetzt auf die nächsten Wochen der Dokumentation und des Erklärens, ich fühle mich um 20 Jahre in die Vergangenheit gebeamt!

Für das App-Projekt „KennzeichenDE“ brauchte ich eine xml-Datei, aus der die SQLite Tabelle `kennzeichen` bestückt wird (mehr Informationen zum Projekt auf der Hompage zu finden). Letztlich habe ich die xml-Datei mit Excel erzeugt, habe mir aber damals schon gedacht, dass muss doch auch mit COBOL gehen und das ist es, was wir jetzt ausprobieren. Der Zeitaufwand ist zwar enorm, aber es macht mir viel Spaß.

Was gibt es sonst noch zu sagen?

Wie gewohnt werde ich in den nachfolgenden Kapiteln sowohl die Installation, als auch die Programmierung selber Schritt für Schritt erklären. Dazu kommen in diesem Falle auch noch diverse Infos rund um COBOL, wie immer in lila gehalten, sie können auch gerne wieder überlesen werden, der Projekterfolg sollte nicht gefährdet sein. Fachliche Vorgaben, die zur Klärung der Anforderung dienen sind in blau gehalten.

Die Sourcen sind kapitelweise als Download verfügbar, ich hoffe das hilft, falls es an der einen oder anderen Stelle mal nicht weiter gehen sollte. Aber wie immer bei der Programmierung und so auch hier gilt – Probieren geht über Studieren.

Bei der Namensgebung für Dateien, Variablen usw. habe ich mich bemüht, Begriffe in deutscher Sprache zu verwenden. Dies hauptsächlich um eine Abgrenzung zu dem in englischer Sprache gehaltenen COBOL-Vokabular zu erreichen. Das gestaltet sich immer dann als schwierig, wenn englische Begriffe wie „SECTION“ mit deutschen Worten wie „DATEI LESEN“ gemischt werden müssen. Sollten mir daher englische oder sogar „denglische“ Begriffe durchgerutscht sein, bitte ich um Nachsicht.

2. Das Projekt

Wie im Vorwort erwähnt, geht es um die „Umformatierung“ von Daten, die in einer Datei im csv-Format vorliegen, in Daten, die dann im xml-Format gespeichert werden. Und das durch ein selbstgeschriebenes COBOL-Programm.

Das war es eigentlich schon, mehr ist dazu von der Anforderungsseite nicht zu sagen.

Wir können aber an dieser Stelle kurz die 3 Begriffe „csv“, „xml“ und „COBOL“ klären.

„**csv**“ steht für „comma seperated values“, frei übersetzt sowas wie „Inhalte, die durch Kommata getrennt sind“. Schauen wir uns das für unseren Fall an, wir haben hier die Datei kennzeichen.csv. Die ersten 3 Zeilen der Datei sehen in einem Text-Editor wie zum Beispiel dem Notepad++ so aus:

```
Abk.;Stadt/Landkreis;abgeleitet von;Bundesland
A;Stadt und Landkreis Augsburg;Augsburg;Bayern
AA;Ostalbkreis;AAlen;Baden-Württemberg
```

Man sieht, dass das Trennkennzeichen eigentlich kein Komma ist, sondern ein Semikolon. In unserem Fall ist das aber okay. Wenn wir das Dokument in einem Tabellenkalkulationsprogramm öffnen, dann fallen die Semikola weg:

	A	B	C	D
1	Abk.	Stadt/Landkreis	abgeleitet von	Bundesland
2	A	Stadt und Landkreis Augsburg	Augsburg	Bayern
3	AA	Ostalbkreis	AAlen	Baden-Württemberg
4				

„xml“ steht für „extensible markup language“, eine Übersetzung ist nicht ganz einfach, „erweiterbare Auszeichnungssprache“ sagt einem erstmal nicht viel. Was bedeutet es aber? Schauen wir uns die Zeilen mal in der xml-Version an:

```
 1 <?xml version="1.0"?>
 2 <kennzeichen>
 3     <record>
 4         <Abk>A</Abk>
 5         <Stadt_Landkreis>Stadt und Landkreis Augsburg</Stadt_Landkreis>
 6         <abgeleitet_von>Augsburg</abgeleitet_von>
 7         <Bundesland>Bayern</Bundesland>
 8     </record>
 9     <record>
10         <Abk>AA</Abk>
11         <Stadt_Landkreis>Ostalbkreis</Stadt_Landkreis>
12         <abgeleitet_von>AAlen</abgeleitet_von>
13         <Bundesland>Baden-Württemberg</Bundesland>
14     </record>
15 </kennzeichen>
```

Was sehen wir da? Schauen wir uns zunächst die Zeilen 4 bis 7 an.

Zeile 4 beginnt mit der Überschrift, die in der Tabelle weiter oben in Zelle A1 steht (<Abk>), nur ohne den Punkt. Dann kommt der Inhalt Zelle A2 (also „A“), gefolgt erneut von der Überschrift Zelle A1, aber mit einem Schrägstrich („/“) vor der Überschrift (</Abk>).

Zeile 5 ist analog, hier ist der Anfang die Überschrift aus Zelle B1 (<Stadt_Landkreis>), allerdings ohne den Schrägstrich, dafür mit Unterstrich. Gefolgt vom Inhalt der Zelle B2 (Augsburg) und dann wieder die Überschrift mit dem Schrägstrich (</Stadt_Landkreis>).

Zeilen 6 und 7 laufen nach dem gleichen Muster ab, das muss also System haben. Hat es auch. Statt „markup“ hat sich der englische Begriff „tag“, also „Etikett“ oder „Schild“ aber eben auch „Marke“ durchgesetzt. Jetzt fängt mein Dilemma an, was ist der Artikel für „tag“? Der tag? Die tag? Oder das tag? Ich entscheide mich für das Neutrum. Ein tag fängt also mit dem Kleiner-Zeichen („<“) an, gefolgt von einem (!) Wort, also ohne Leerzeichen, den Abschluss bildet das Größer-Zeichen („>“).

Unterschieden werden öffnende und schließende tags. Ein Beispiel für ein öffnendes tag ist unsere Überschrift <Abk>, das schließende tag ist also </Abk>. Alles was dazwischen steht wird als eine Einheit betrachtet.

Schauen wir uns jetzt die Zeilen 3 und 8 an, auch sie bilden ein tag-Paar mit <record> und </record>. Das tag-Paar bildet also die logische Klammer um einen Datensatz, also eine Zeile in der oben abgebildeten Tabelle. Die Überschriften der Zellen sind also implizit durch die öffnenden und schließenden tags mit enthalten. Cool oder?

Die Zeilen 9 bis 14 bilden also den Inhalt der 2. Zeile in der obigen Tabelle ab. Die Zeilen 2 und 15 umschließen wiederum alle Datensätze der Tabelle. Einfach, oder?

Zum Schluss noch zu Zeile 1, dies sind die Steueranweisungen für den Umgang mit dieser Datei. Hier können auch Definitionen wie der verwendete Zeichensatz, z.B. „UTF-8“ abgelegt werden. Uns reichen die Angaben oben aus, damit weiß jedes Programm, dass es sich hierbei um eine xml-Datei handelt.

Zum letzten Begriff „**COBOL**“. Mich hier kurz zu fassen ist nicht einfach, ich versuche es trotzdem. COBOL steht für „**CO**mmon **B**usiness **O**riented **L**anguage“, frei übersetzt so etwas wie „allgemeine, sich an der Betriebswirtschaft orientierende (Programmier-)Sprache“. Es gibt einen sehr guten Artikel in Wikipedia dazu, wer sich dafür interessiert kann sich das gerne mal durchlesen.

Um die immer größer werdenden Datenmengen in den Griff zu bekommen, gab das US-Verteidigungsministerium in den 1950er Jahren den Auftrag, eine neue Programmiersprache zu entwickeln, die sich eben um die schnelle Verarbeitung großer Datenmengen kümmern sollte.

Heraus kam 1960 „COBOL-60“, die Speicherung des Quellcodes erfolgte auf Lochkarten, weshalb die feste Spaltenzuordnung im Quellcode bis heute noch eingehalten werden muss. Kommen wir aber später dazu.

Seit 1960 hat sich natürlich viel getan und die Sprache ist immer wieder erweitert worden. Was sich allerdings nicht geändert hat, COBOL ist eine „Compiler-Sprache“, wir brauchen also einen Compiler oder „Übersetzer“, der uns den von uns geschriebenen Quellcode in Maschinencode übersetzt.

Bis sich die Gemeinschaft der Open-Sourcler in 2002 an die Entwicklung eines freien Compilers gemacht hat, wäre es und nicht möglich gewesen, dieses Projekt hier zu starten. Daher haben wir bei den Vorarbeiten noch einiges zu tun, aber das kriegen wir hin. Auf geht’s!

3. Vorarbeiten

Programmiert wird in der freien IDE OpenCobolIDE, die letzte mir bekannte Version ist die 4.7.6.

Das fertig entwickelte Programm dann aber zu starten, ist nicht ganz so einfach. Wir erinnern uns, der programmierte Code muss in einen von einer Maschine lesbaren Code übersetzt werden, das sogenannte „Kompilieren“. Für diese Übersetzungsleistung benötigen wir also einen COBOL-Compiler. In der IDE ist ein Compiler integriert, auch eine Laufzeitumgebung, allerdings nicht für umfangreiche Module.

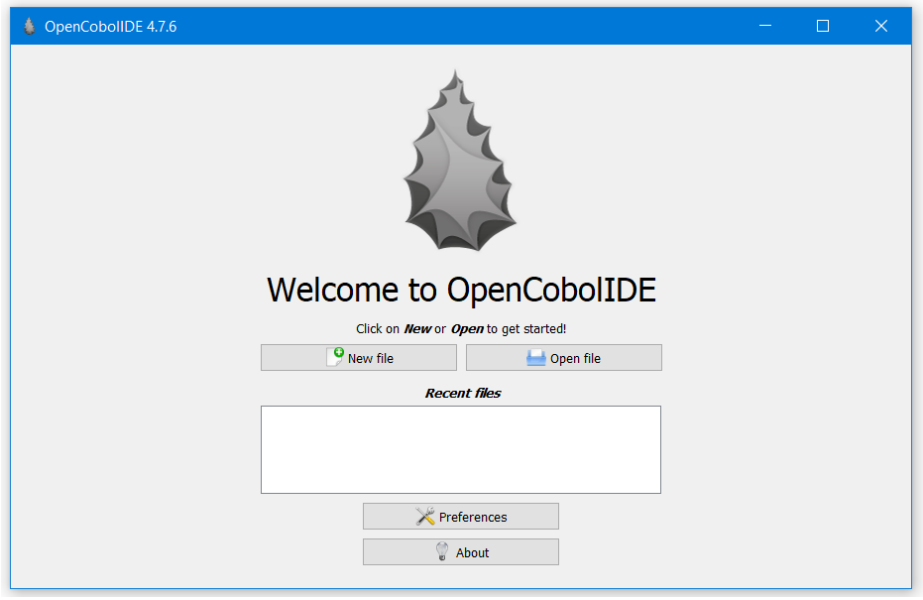
Da ich unter Windows arbeite, muss das fertige Produkt eine exe-Datei sein, sonst können wir es nicht unter Windows starten. Möglich gemacht wird das, da der Quellcode zuerst in ein C-Programm umgewandelt wird, dieses dann wieder in eine unter Windows ausführbare Datei.

Windows selbst stellt uns für das Kompilieren keine komfortable Möglichkeit zur Verfügung, wir müssen uns daher eine Alternative suchen. Die gibt es zum Beispiel in der Gestalt von Cygwin. Cygwin ist ein Programm, das eine Unix-Umgebung innerhalb von Windows zur Verfügung stellt, mit einer eigenen Konsole, in der alle Befehle zum Kompilieren und Starten ausgeführt werden.

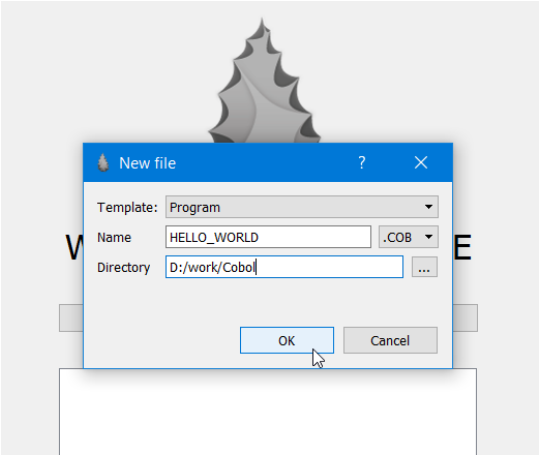
3.1. Die OpenCobolIDE

Zuerst laden wir uns die OpenCobolIDE herunter. Aktuell ist die Version 4.7.6 die neueste, die ich finden konnte. Es gibt diverse Angebote zum Download, ich habe mich für die aus dem GitHub entschieden, zu finden unter https://github.com/OpenCobolIDE/OpenCobolIDE/releases/tag/4.7.6. Auf der Seite unten wird die Datei OpenCobolIDE-4.7.6_Setup.exe bereitgestellt.

Die Installation ist unauffällig, ich habe keine Änderungen an den Vorschlägen des Installations-Assistenten gemacht. Das Startfenster sollte dann so aussehen:

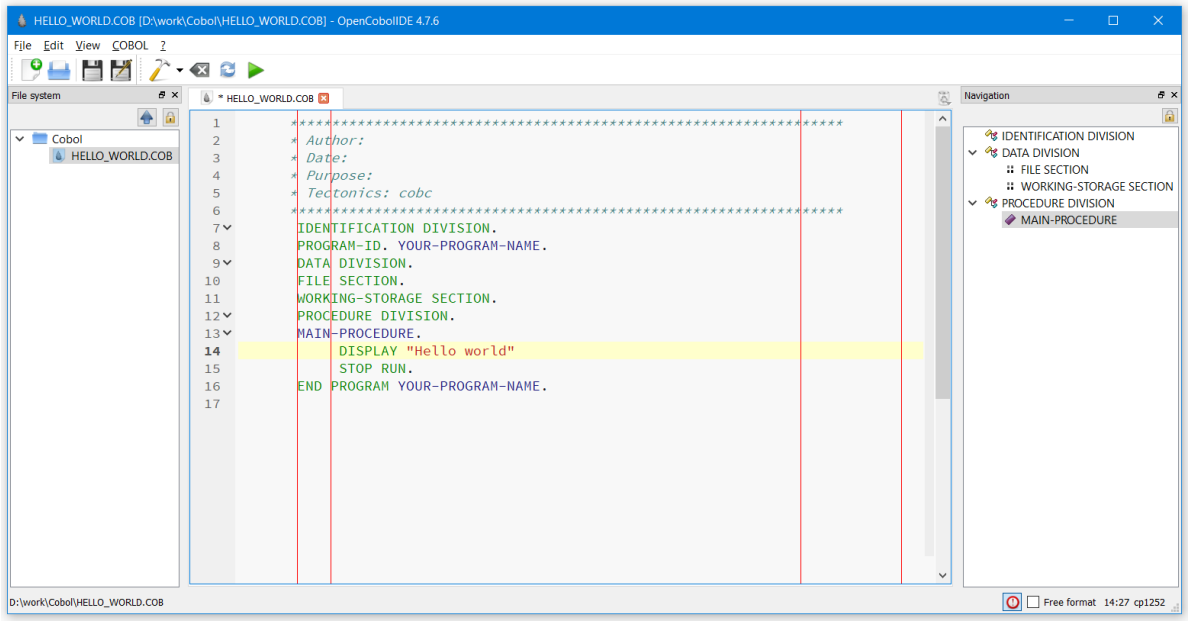


Mit „New File“ öffnet sich ein neues Fenster:



Als Template lassen wir „Program“, vergebt einen Namen und wählt eine Endung aus, ich habe mich für COB entschieden hätte aber auch „cob“ oder „cbl“ nehmen können, ich bin es gewohnt, Großbuchstaben zu verwenden. Den Speicherort auswählen und mit OK bestätigen.

Dann sollten wir nachfolgendes Bild sehen:




```
      ******************************************************************
      * Author:
      * Date:
      * Purpose:
      * Tectonics: cobc
      ******************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. YOUR-PROGRAM-NAME.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
       MAIN-PROCEDURE.
            DISPLAY "Hello world"
            STOP RUN.
       END PROGRAM YOUR-PROGRAM-NAME.
```

Wir sehen 3 Fenster, in der Mitte den Editor zur Bearbeitung des Quelltextes, links die Projektstruktur und rechts die Navigation innerhalb des Quellcodes.

Im Editor sind 4 senkrechte Striche zu sehen. Wie oben erwähnt, orientierte sich COBOL bei der Entwicklung in den 1950er Jahren an der Lochkartencodierung mir 80 Spalten. Einen automatisierten Textumbruch gab es zu der Zeit noch nicht, weshalb auch das Schema für den Editor sehr starr ist. Jeder Abschnitt der Lochkarte hatte seine eigene Bedeutung.

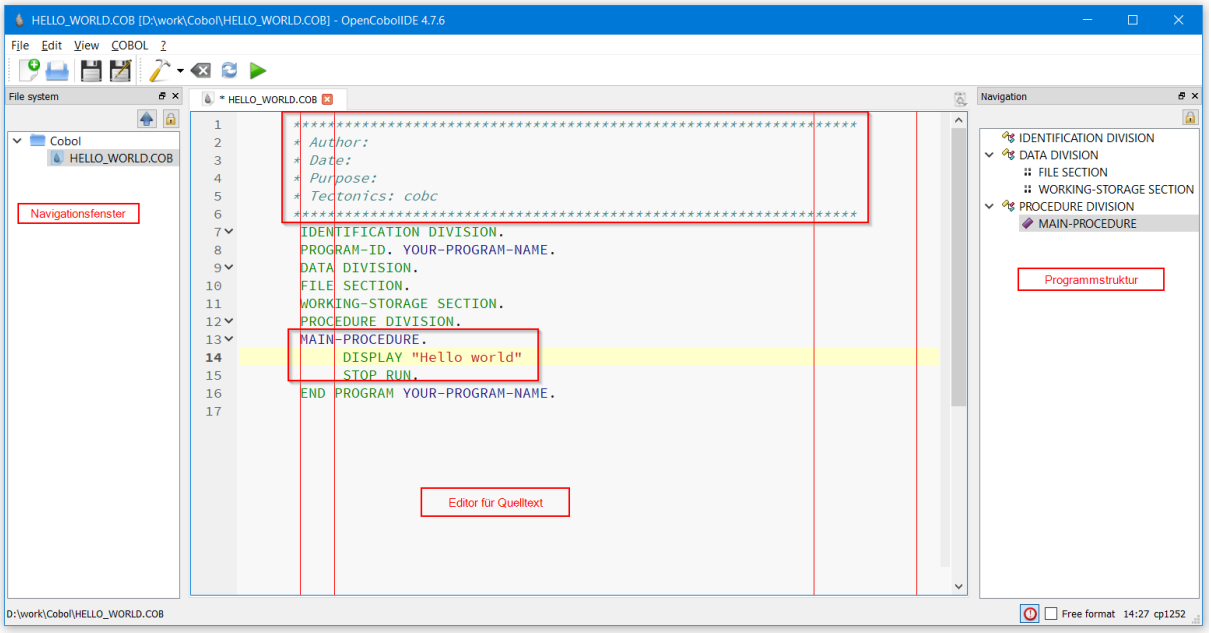
So sind die ersten 6 Spalten Freitext, meistens werden sie als Zeilennummerierung genutzt. In unserem Fall sind sie leer. Steht in Spalte 7 ein Stern („*“) wird der Rest der Zeile als Kommentar interpretiert.

Die nächste Trennung ist zwischen Spalte 11 und 12. Ein COBOL-Programm ist nach einer bestimmten Abfolge ähnlich wie Kapitel-Ebenen aufgebaut. Der Compiler erwartet zum Beispiel die Deklaration

von Daten in der WORKING-STORAGE SECTION, die wiederum nur unterhalb der DATA DIVISION stehen darf. Das ist am Anfang vermutlich verwirrend, wird aber mit der Zeit zur Routine.

Für jetzt nehmen wir mit, dass im Abschnitt der Spalten 8 bis 11 nur Kapitel-Überschriften stehen dürfen.

Der Strich vor Spalte 68 erschließt sich mir nicht, der Strich vor Zeile 80 ist wieder klar, das ist das Ende der Zeile, darüber hinaus darf kein Text stehen.



Zeile 7 ist die erste der `DIVISIONs`. Sie war früher umfangreicher, in der Zwischenzeit ist nur noch die PROGRAMM-ID übergeblieben.

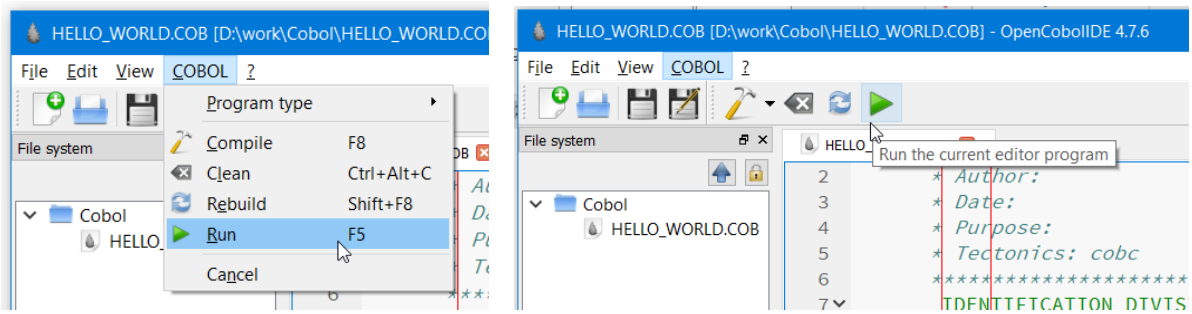
Hier sehen wir auch schon ein weiteres wichtiges Element, den Punkt. Nach dem Wort DIVISION oder SECTION steht immer ein Punkt, alles was danach kommt, sind Anweisungen in diesem Abschnitt, bis zu einem abschließenden Punkt. Im obigen Beispiel haben wir die PROGRAM-ID gefolgt von einem Punkt, dann den Programmnamen, wieder gefolgt von einem Punkt, dieser schließt das Kapitel PROGRAM-ID ab. Ein *vergessener* Punkt hat mich schon Stunden der Suche gekostet.

Zeile 8 korrespondiert mir Zeile 16, die Inhalte müssen identisch sein, sonst meckert de Compiler.

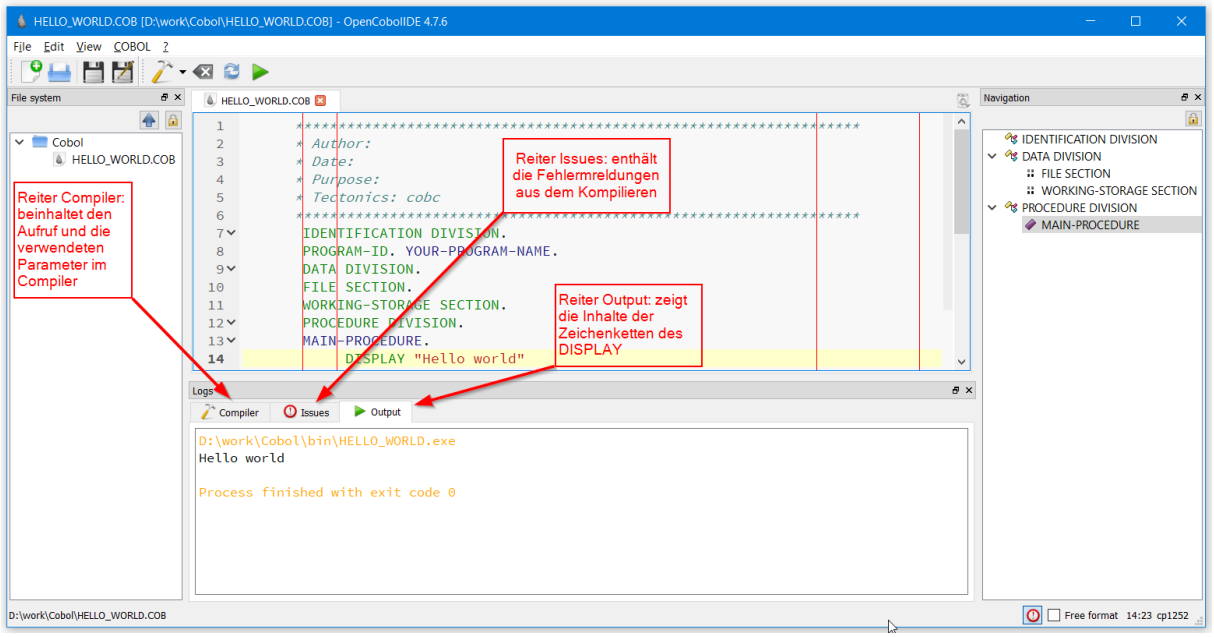
Die PROCEDURE-DIVISON ist die eigentliche Programmverarbeitung. Auch sie endet mit einem Punkt, zur Verdeutlichung, dass jetzt wirklich Schluss ist, steht vor dem Punkt noch das `STOP RUN`.

Die einzige Aktivität, die das Programm macht, ist die Ausgabe der Zeichenkette `Hello world`. Dass es sich hierbei um eine Zeichenkette handelt, sehen wir an den Anführungsstrichen vor und nach der Zeichenkette, also `"Hello world"`.

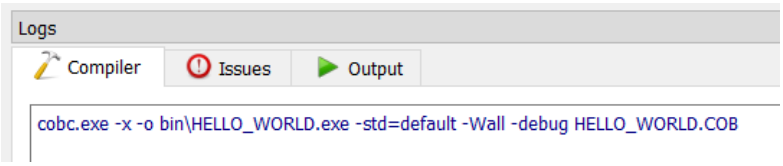
Wir können das auch ausprobieren. Mit COBOL/Run oder F5, oder dem Pfeil-Icon



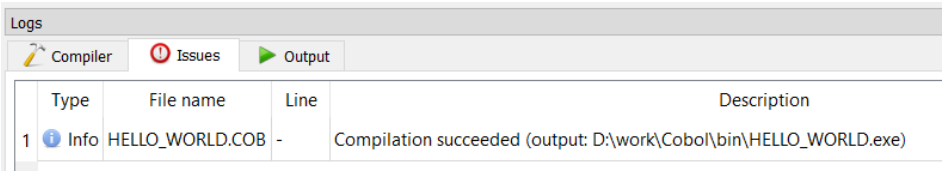
geht ein neues Fenster (Logs) unterhalb des Editors auf und zeigt uns die Ausgabe:



Weiter gibt es noch 2 andere Reiter. „Compiler“ enthält die Parameter der Umwandlung selbst:



Sollte es Fehler gegeben haben, werden die hier angezeigt:



Für dieses kleine Programm kann man die Run-Funktionalität nutzen, für etwas komplexere Module geht das nicht mehr. Daher wenden wir uns gleich Cygwin zu, da ist das kein Problem.

3.2. Cygwin

Jetzt wird es etwas schwieriger, aber auch das kriegen wir hin.

Wie beschrieben stellt Cygwin uns eine Unix-artige Umgebung zur Verfügung. Es ist sozusagen eine Hülle mit einer Konsole, in die wir weitere Module in Form von Paketen einlagern müssen. Wie, das sehen wir gleich.

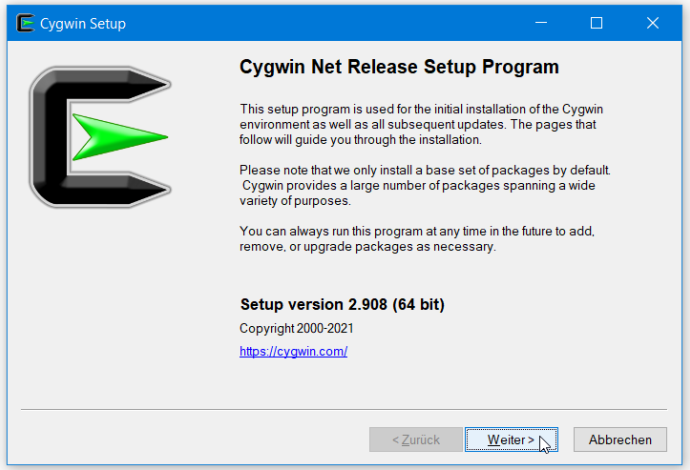
Ich selbst habe mich bei der Installation auf die Internetseite

https://www.it-cooking.com/projects/how-to-install-gnucobol-for-cygwin/

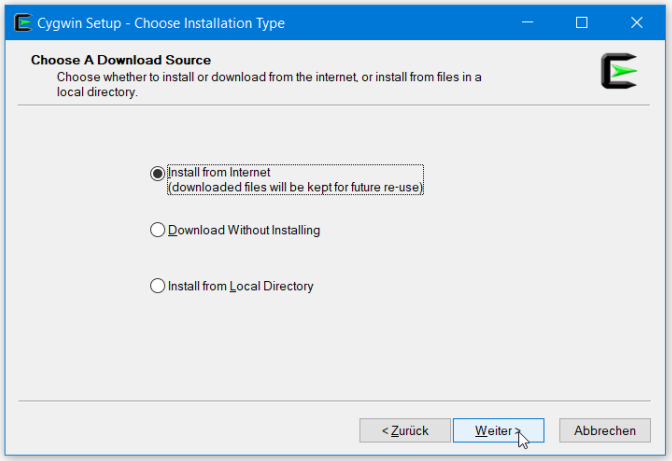
verlassen. Hier sind alle notwendigen Schritte beschrieben, auch wenn die enthaltenen Versionen für einige Pakete nicht mehr aktuell sind. Und die Berkeley Datenbank habe ich auch erst nach weiterer Recherche gefunden, dazu aber gleich mehr.

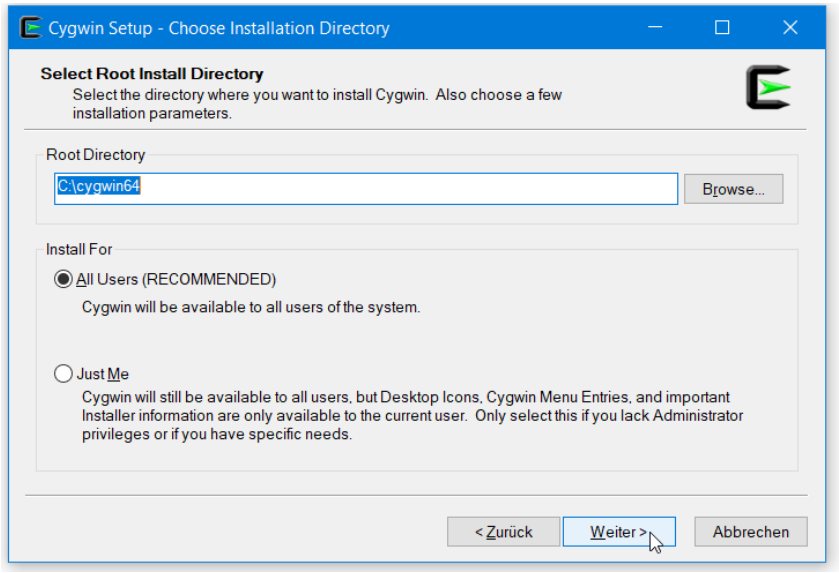
Los geht es mit dem Herunterladen der setup-x86_64.exe von Cygwin, diese ist zu finden auf https://www.cygwin.com/install.html.

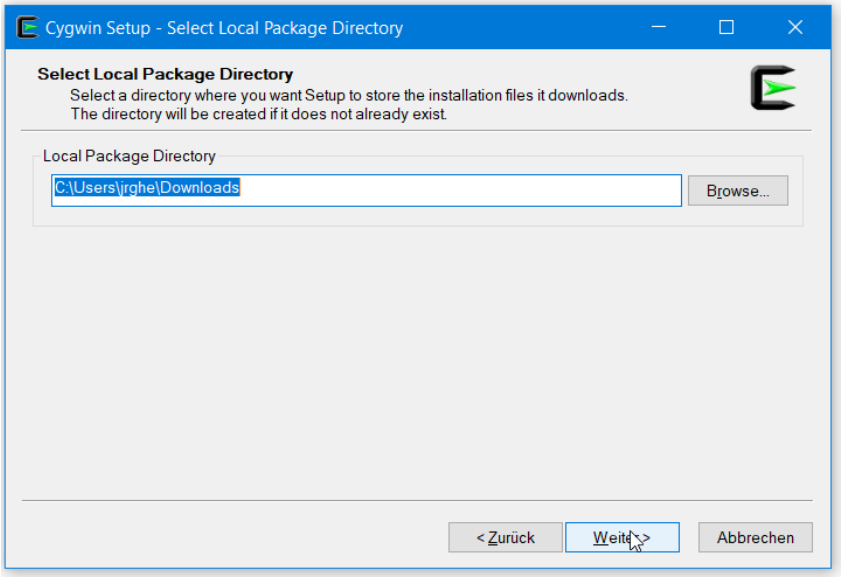
Die Datei brauchen wir immer wieder, wenn wir Pakete dazuladen wollen. Nach Ausführung der exe erscheint bei mir die Installation der Version 2.908:

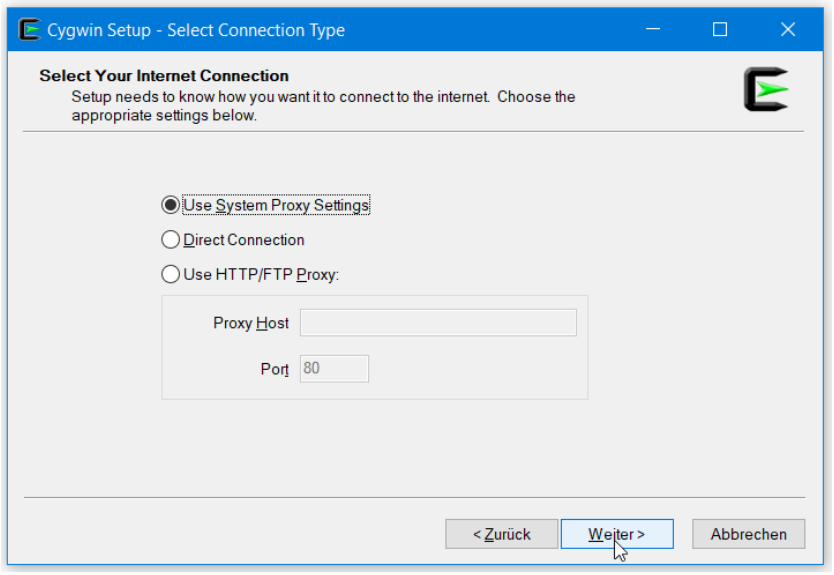


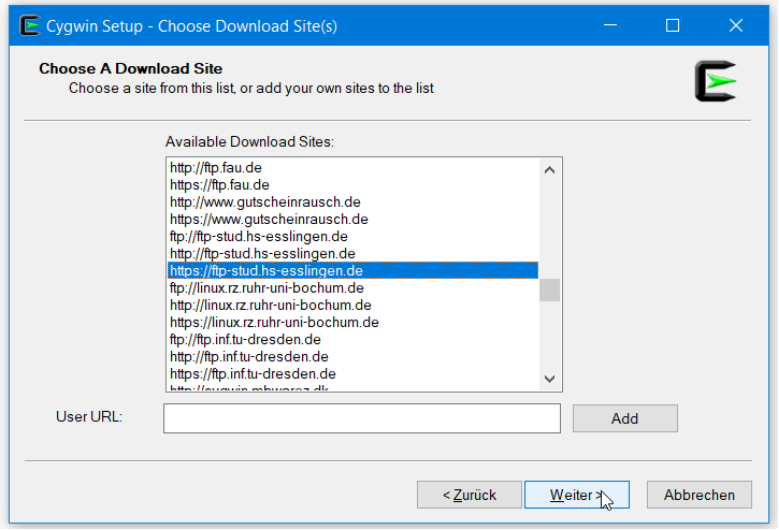
Die nächsten Screenshots lasse ich unkommentiert mal drin:



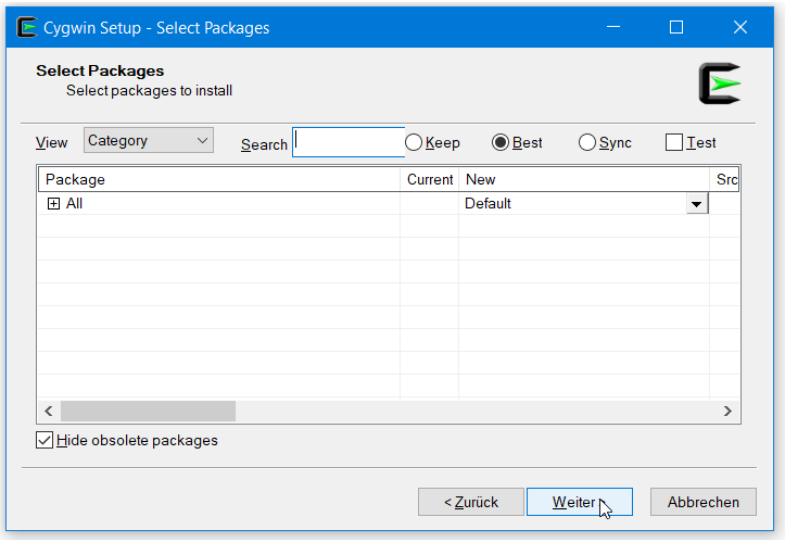

Cygwin Setup - Choose Installation Directory
Select Root Install Directory
Select the directory where you want to install Cygwin. Also choose a few installation parameters.
Root Directory
C:\cygwin64
Browse...
Install For
All Users (RECOMMENDED)
Cygwin will be available to all users of the system.
Just Me
Cygwin will still be available to all users, but Desktop Icons, Cygwin Menu Entries, and important Installer information are only available to the current user. Only select this if you lack Administrator privileges or if you have specific needs.
< Zurück
Weiter >
Abbrechen

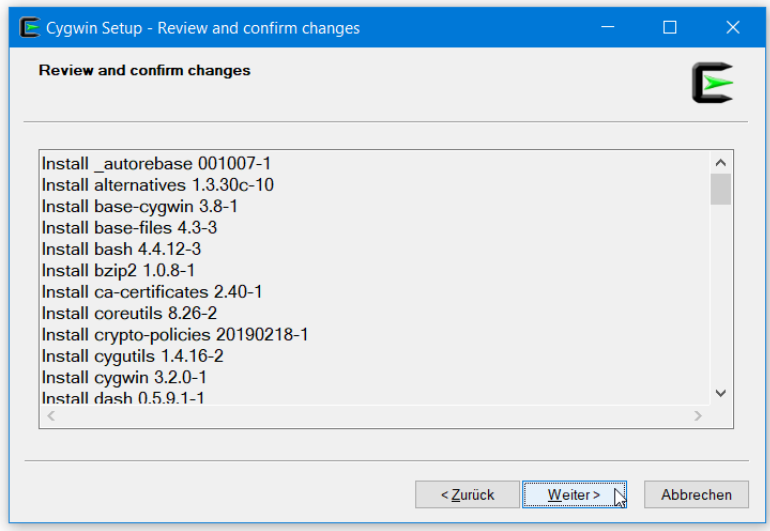

Cygwin Setup - Select Local Package Directory
Select Local Package Directory
Select a directory where you want Setup to store the installation files it downloads. The directory will be created if it does not already exist.
Local Package Directory
C:\Users\jrghe\Downloads
Browse...
< Zurück
Weiter >
Abbrechen

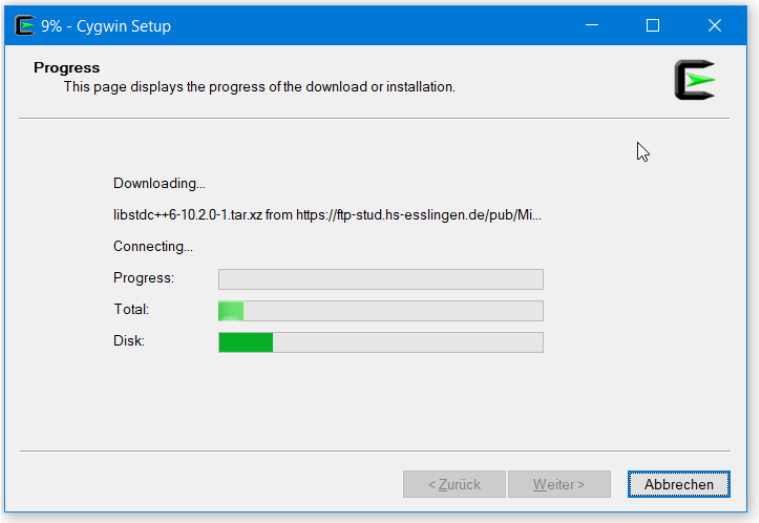

Cygwin Setup - Select Connection Type
Select Your Internet Connection
Setup needs to know how you want it to connect to the internet. Choose the appropriate settings below.
Use System Proxy Settings
Direct Connection
Use HTTP/FTP Proxy:
Proxy Host
Port
80
< Zurück
Weiter >
Abbrechen

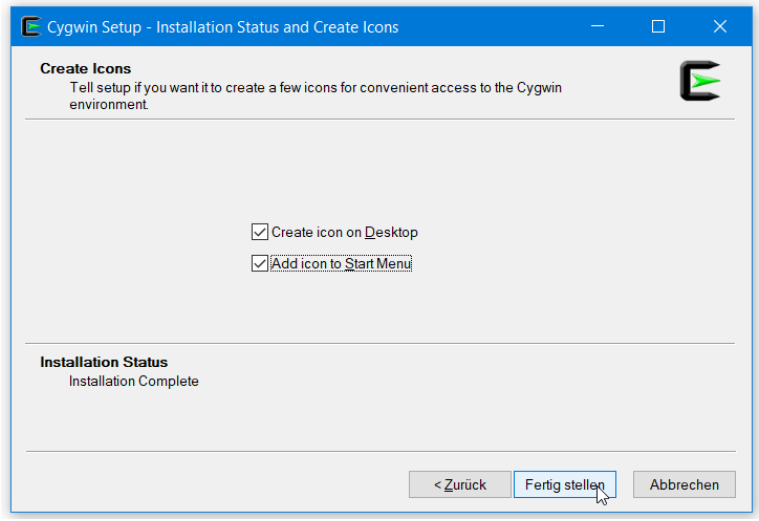


Hier könnt Ihr Euch jeden beliebigen Server aussuchen.

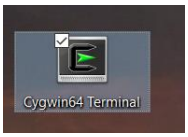




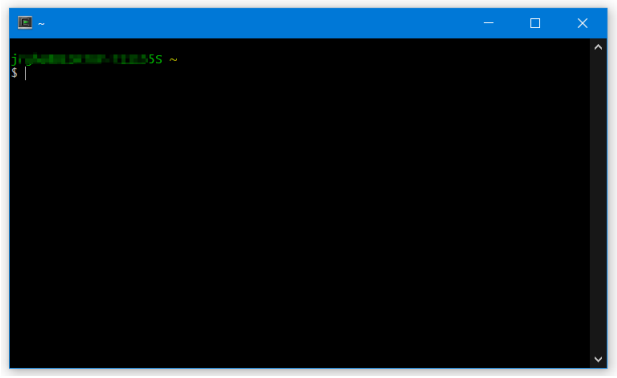




Damit haben wir quasi die Basis-Version, also die Hülle angelegt. Wir können das jetzt ausprobieren, wir sollten auf dem Desktop ein Icon haben:

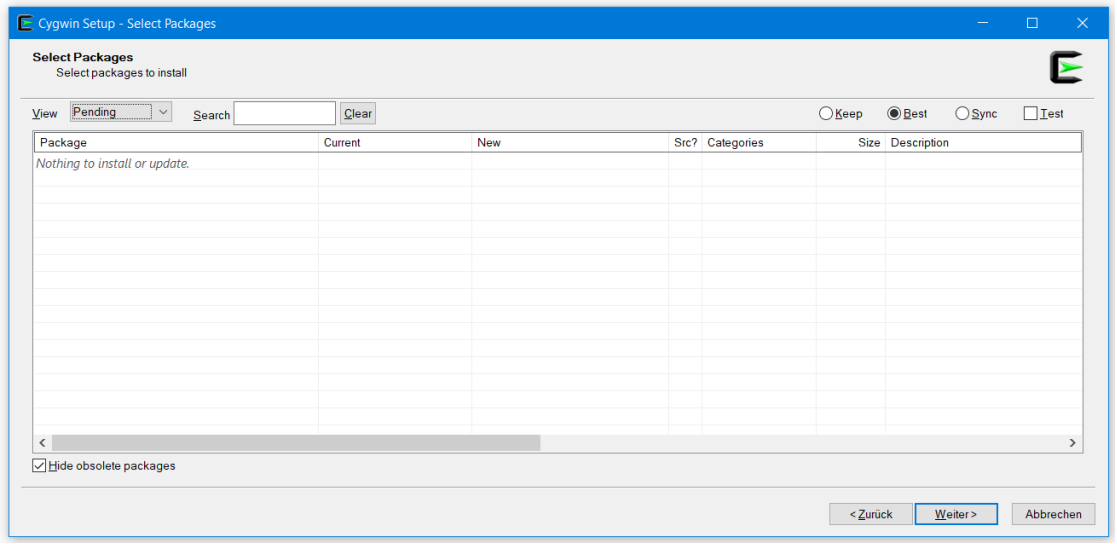


Das klicken wir an. Nun öffnet sich das Terminalfenster:

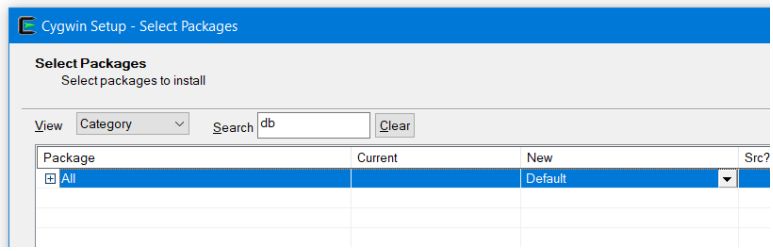


Verlassen können wir das Fenster, indem wir „exit“ eingeben.

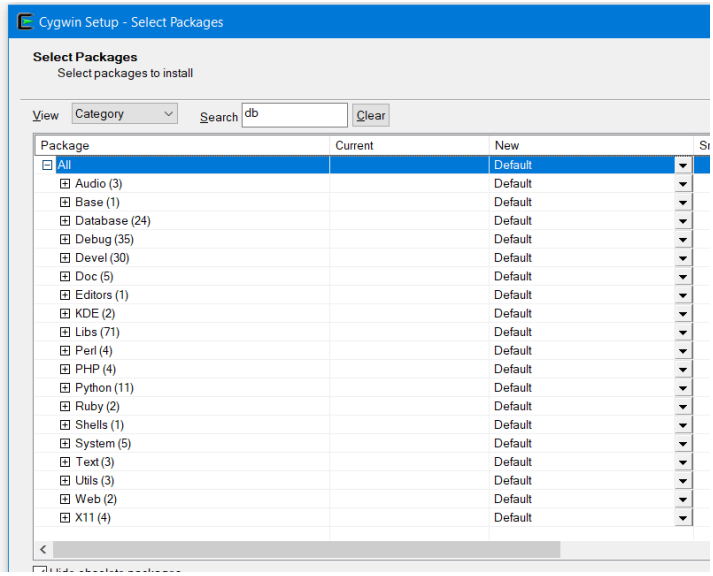
Jetzt fängt das Zusammensammeln der benötigten Pakete an. Wenn wir erneut das Setup-Programm aufrufen landen wir nach mehreren Klicks wieder in der folgenden Ansicht:



Das sieht nachfolgend etwas anders aus als in der online-Hilfe, deshalb hier die ersten Schritte mit aktuelleren Screenshots. Wählen wir „Category“ im Drop-Down für View aus und geben in Search „db“ ein, sehen wir:

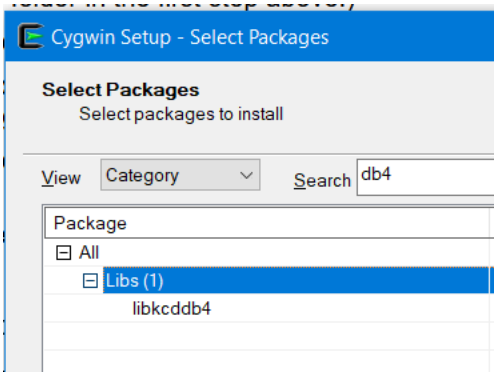


Klappen wir das durch Klick auf das Pluszeichen neben „All“ auf, werden wir schier erschlagen:

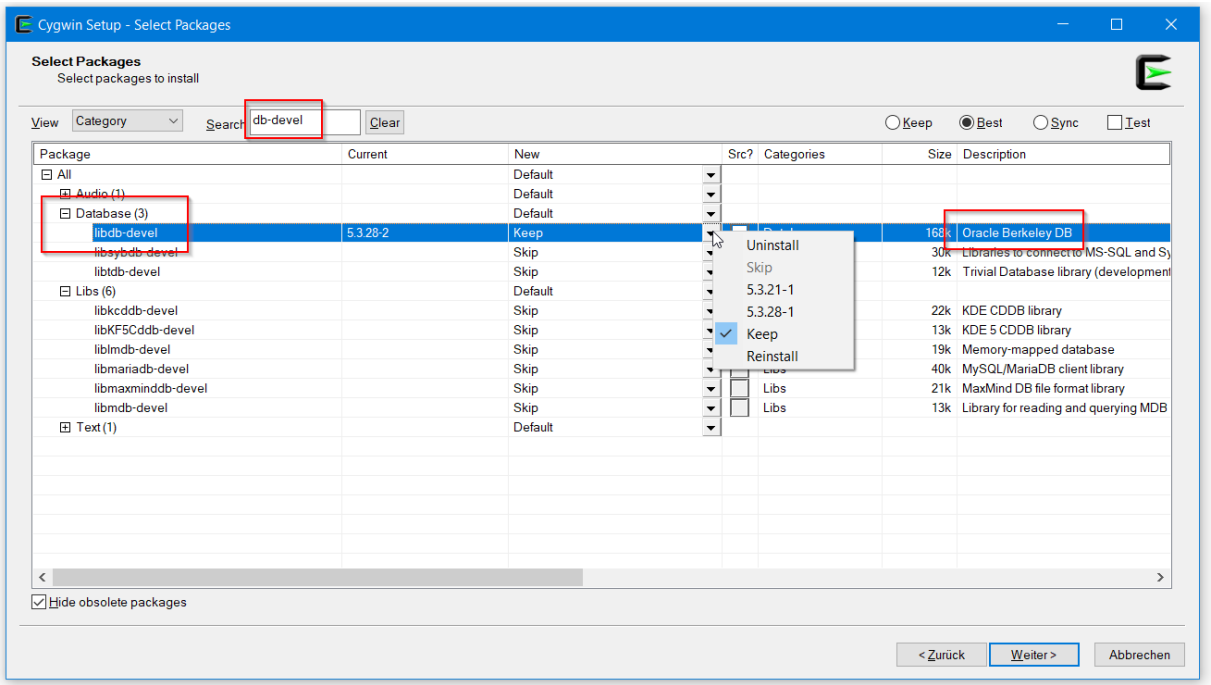


Sinn der Sache ist, genau die in der online-Hilfe geforderten Pakete zu finden und zu installieren.

Der erste Abschnitt beschreibt die Datenbanken. Ich habe nach „db4“ gesucht aber da war nichts:



Bei der Installation des Compilers ist es dann zur Fehlermeldung gekommen „Berkeley DB not installed“. Im Internet habe ich dann den Hinweis auf das Paket „db-devel gefunden“. Das habe ich dann in Cygwin in den Search gegeben und in der Gruppe Database bin ich fündig geworden:

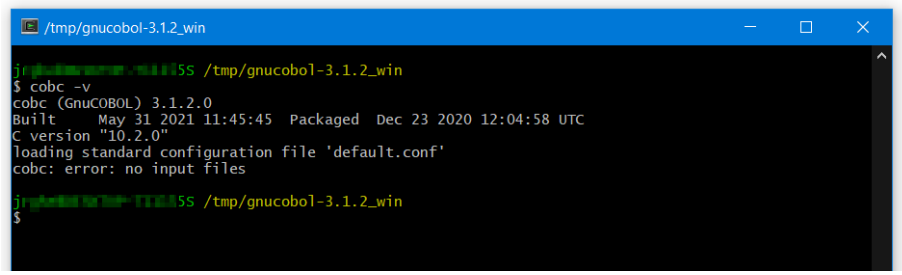


Wie man sieht unterscheidet sich die aktuelle Version Cygwin leicht von der in der online-Hilfe.

Das Prinzip ist aber gleich. Ist in der Spalte „Current“ eine Versionsnummer enthalten, ist diese installiert. Ist die Spalte leer, ist eben nichts installiert. Die Installation selbst geht dann über das Drop-Down in der Spalte „New“. Klickt man eine Versionsnummer an, wird diese nach dem Klick auf „Weiter“ installiert.

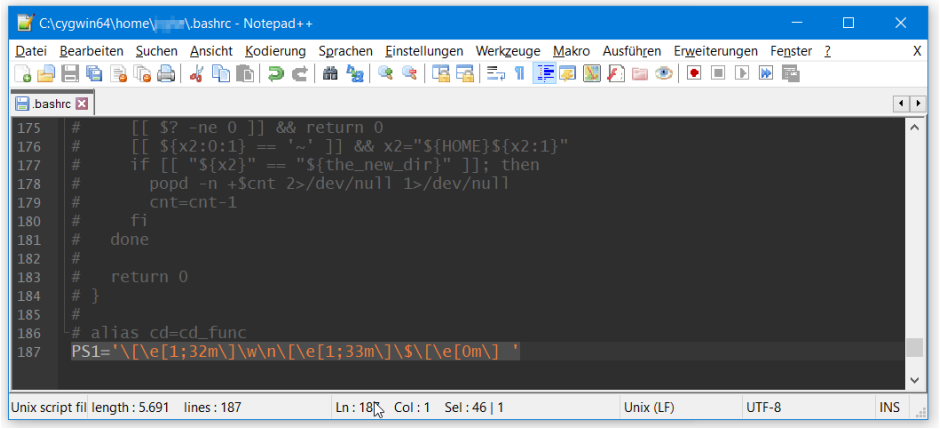
Das war es auch an Besonderheiten bei der Installation von Cygwin. Der nächste Punkt ist dann die Installation des Compilers, auch das ging der Anleitung folgend einwandfrei, allerdings nicht mit der Version 2.2 des gnuCOBOL, sondern Version 4.7.6.

Nach Aufruf von „cobc -v“ sehe ich den folgenden Inhalt:

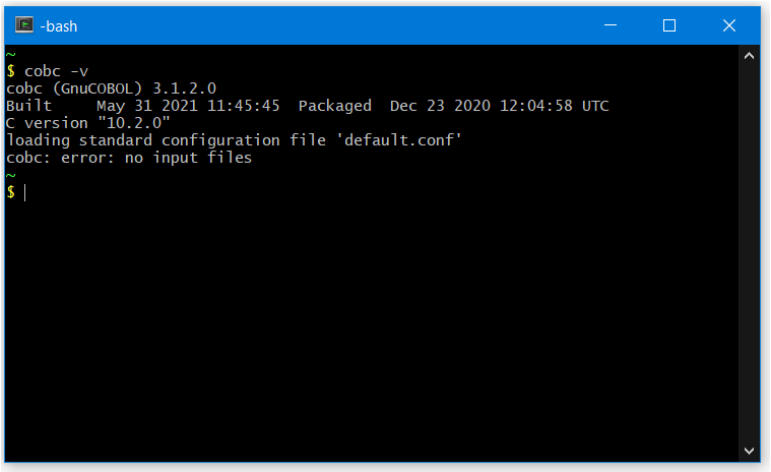


So sollte es sein. Falls das bei Euch anders aussieht, geht nochmal die Pakete durch, vielleicht ist eines davon noch nicht installiert.

Mich stört der angezeigte Nutzer- sowie der Rechner-Name, das muss ich ändern. Die Steuerung der Konsole erfolgt über das Programm „bash“. Die Anzeige darin über das Argument „PS1“. Bei Aufruf des Terminals werden die Informationen aus der Datei „bashrc“ auf Eurem Benutzerordner c:\cygwin64\home\<user> gezogen. Wenn Ihr die Datei mit einem Text-Editor (in meinem Fall Notepad++) öffnet, könnt Ihr ganz ans Ende gehen und eine neue Zeile einfügen:



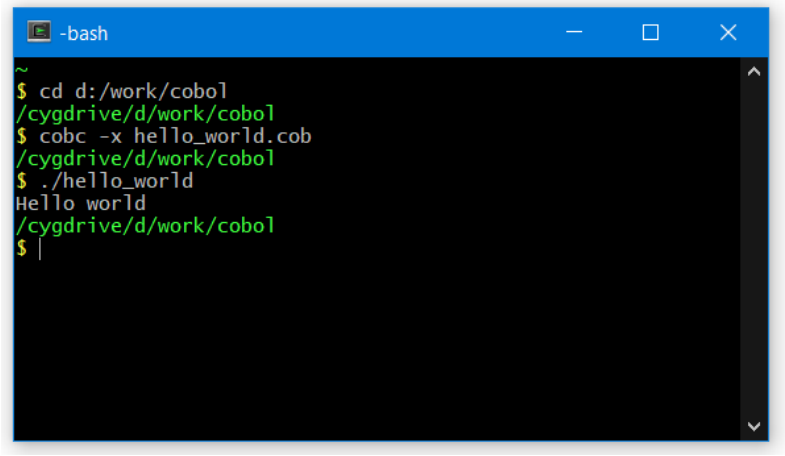
Der Text dafür ist `PS1='\[\e[1;32m\]\w\n\[\e[1;33m\]\$\[\e[0m\] '` . Auch damit könnt Ihr je nach Lust und Laune rumspielen. Gefällt Euch nichts davon, Zeile löschen und alles ist beim Alten. Die Einstellungen ergeben dann folgendes Bild:



Jetzt aber zu unserem Testprogramm HALLO_WORLD.COB. Wir müssen zunächst in den Ordner navigieren, in dem die Datei liegt. Bei mir ist das D:\work\Cobol\

Um dahin zu kommen, gebe ich „cd d:/work/cobol“ ein, zu beachten die Richtung der Schrägstriche ändert sich.

Danach „cobc -x hello_world.cob“, was uns die exe-Datei generiert. Der Aufruf der exe erfolgt mit „./hello_world“, also ohne die Endung. Die Display-Zeile erscheint dann unterhalb des Aufrufs:



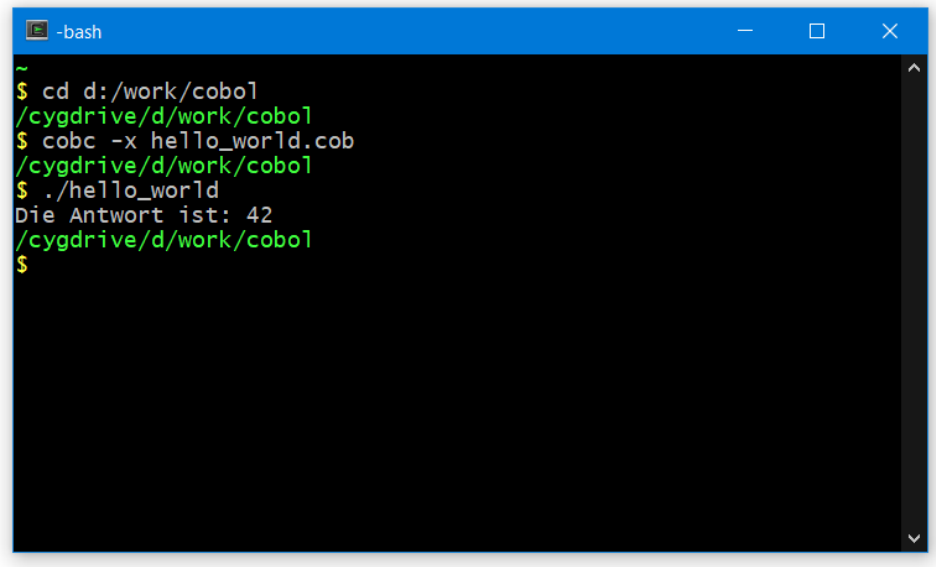
Super, damit haben wir unsere Arbeitsumgebung zusammen. Programmieren in der DIE, Umwandeln und Ausführen in Cygwin.

Um das Zusammenspiel noch einmal zu testen, ändern wir schnell einmal die Ausgabe. Der neue Quellcode lautet:

```
1        * Author:
2        * Date:
3        * Purpose:
4        * Tectonics: cobc
5        ******************************************************************
6         IDENTIFICATION DIVISION.
7         PROGRAM-ID. YOUR-PROGRAM-NAME.
8         DATA DIVISION.
9         FILE SECTION.
10        WORKING-STORAGE SECTION.
11        01  FRAGE.
12            05 FRAGE_NACH_ALLEM PIC X(19) VALUE
13            "Die Antwort ist: 42".
14        PROCEDURE DIVISION.
15        MAIN-PROCEDURE.
16             DISPLAY FRAGE_NACH_ALLEM
17             STOP RUN.
18        END PROGRAM YOUR-PROGRAM-NAME.
```

Hier bauen wir unsere erste „Variable“ ein, und geben diese aus.

Bei mir ist das dann:



Mit der Syntax befassen wir uns im Laufe des Programms ausführlich, hier ging es mir erstmal darum, dass wir den korrekten Aufbau der Arbeitsumgebung haben. Das haben wir geschafft, jetzt geht es an die Programmierung.

Im Zweifel habe ich das Manual https://gnucobol.sourceforge.io/doc/gnucobol.pdf gerne zur Hand.

4. Projekt „csv nach xml“

Was wollen wir tun? Wir haben eine csv-Datei in der die Informationen zu den Kennzeichen stehen und wollen diese in eine xml-Datei umformatieren.

Bevor es aber losgeht, sollten wir uns noch ein paar allgemeine Gedanken machen, damit wir uns nachher einfacher zurechtfinden.

Die nachfolgenden Kapitel sind so aufgebaut, dass am Ende ein lauffähiges Artefakt herauskommt, das im Download-Bereich zur Verfügung steht.

4.1. Die Programmstruktur

COBOL folgt dem „imperativen Programmierparadigma“. Den Imperativ kennen wir aus der Schule – das ist die Befehlsform. Also „räum Dein Zimmer auf!“ oder „bring den Müll weg!“. In unserem Beispiel-Programm ist das „Gib den Inhalt der Variablen FRAGE_NACH_ALLEM auf der Konsole aus“, zu finden Zeile 16. Damit ist also gemeint, ein Programm ist eine Ansammlung von Befehlen oder ANweisungen die in der im Programm festgelegten Reihenfolge abgearbeitet werden.

Die Organisation im Programm selbst ist meistens prozedural. Das wiederum soll heißen, dass Programmcode in kleine – auch gerne wiederverwendbare – Prozeduren geteilt wird, damit der Code les- und wartbar ist. Darauf sollte man immer dann achten, wenn die Möglichkeit besteht, dass jemand anderes den Code erweitern muss. Aber auch für einen selbst ist das interessant, wenn man nach Monaten oder gar Jahren seinen eigenen Code wieder lesen muss. Daher gibt es in den meisten Unternehmen die COBOL einsetzen, Richtlinien zur Gestaltung des Codes. Schauen wir uns das genauer an.

Der ausführbare Code fängt nach dem Punkt in der PROCEDURE DIVISION an. Die letzte Anweisung ist END PROGRAM YOUR-PROGRAM-NAME. – strenggenommen ist der letzte Punkt.

Das was dazwischen steht, ist unser Code. Wie wir ihn organisieren, ist uns überlassen.

In unserem Beispiel-Programm ist die erste Anweisung nach der PROCEDURE DIVISION der Aufruf der MAIN-PROCEDURE.

Wenn es eine Haupt-PROCEDURE gibt, muss es doch auch Unter-PROCEDUREs geben, oder? Richtig, die gibt es, das sind die SECTIONs. In der Haupt-PROCEDURE wird der Ablauf des Programms festgelegt, es ist sozusagen unser Inhaltsverzeichnis des Programms. Um in eine SECTION zu gelangen, erfolgt ein Aufruf mit dem Befehl PERFORM section-name. Sobald die SECTION beendet ist, landet man wieder hinter dem PERFORM.

Klingt kompliziert, ist es aber nicht, das sehen wir gleich.

Rufen wir uns in Erinnerung, was wir machen wollen. Wir haben eine Eingabe (csv-Datei) und eine Ausgabe (xml-Datei) und müssen die Daten auf dem Weg von einer zur anderen Datei neu strukturieren. Was brauchen wir aber alles, damit unser Programm funktioniert?

Zuerst müssen wir uns die Ausgangslage für das Programm schaffen. Das ist die **Initialisierung**. Danach müssen wir die beiden Dateien zur **Ein- und Ausgabe öffnen**. Im Anschluss **lesen** wir **zeilenweise** die csv-Datei, **konvertieren** jeden csv-Satz in einen xml-Satz und **schreiben** ihn in die xml-Datei. Das ist also unsere **Verarbeitungsschleife**.

Nachdem wir den letzten Satz verarbeitet haben, müssen wir die beiden **Dateien** noch **schließen** und dann sind wir fertig.

Mein Vorschlag für eine Reihenfolge ist

- Beginne mit der STEUER-SECTION
- Rufe Initialisierung auf
- Öffne die beiden Dateien
- Lies den erster Datensatz
- Verarbeite, solange Sätze in der csv-Datei sind oder ein Fehler aufgetreten ist
  - Prüfe, ob das Ende der Datei erreicht oder ein Fehler aufgetreten ist
    - Wenn eins von beiden zutrifft, dann beende die Verarbeitung
    - Sonst gehe in die Konvertierung
      - ❖ Prüfe dort zuerst, ob es der erste Datensatz ist
        - ➢ Wenn ja, schreibe Zeile 1 und Zeile 2 in die xml-Datei
        - ➢ Wenn nein, dann konvertiere Objekt für Objekt von csv nach xml und schreibe Zeile für Zeile in xml-Datei
    - Lies den nächsten Datensatz
- Führe Abschlussarbeiten durch
- Schließe die beiden Dateien
- Beende die STEUER-SECTION
- Beende das Programm

So weit so einfach. Gehen wir es durch und machen uns erste Gedanken zur Kodierung:

Aktion	**Umsetzung und COBOL-Code**
Beginne mit der STEUER-SECTION	Das ist einfach: `STEUER SECTION.`

Rufe Initialisierung auf	Der Aufruf einer anderen SECTION erfolgt durch PERFORM section-name. In unserem Fall also
	
	`PERFORM INITIALISIERUNG`
	
	Wichtig hier – kein Punkt und auch kein „SECTION“ hinterher.
	
	Zusätzlich brauchen wir außerhalb (also nach der END PROGRAM YOUR-PROGRAM-NAME-Zeile) eine neue SECTION:
	
	`INITIALISIERUNG SECTION.`
	`.`

	Richtig, mit 2 Punkten, einen direkt nach SECTION und einen am Ende der SECTION, damit der Compiler weiß, dass hier das Ende der SECTION ist und wieder in die STEUER SECTION zurückspringt.

Öffne die beiden Dateien	Jetzt brauchen wir etwas mehr. Erstmal machen wir es uns einfach und bauen nur den Aufruf der SECTION in die STEUER SECTION ein, und wie bei der Initialisierung auch die SECTION selbst. Also der Aufruf:
	
	`PERFORM OEFFNEN-DATEIEN`
	
	Und die SECTION selbst:
	
	`OEFFNEN-DATEIEN SECTION.`
	`.`
	
	Damit sind wir noch nicht fertig, aber die Struktur passt.

Lies den erster Datensatz	Da wir das Lesen der csv-Datei mehrfach durchführen, machen wir das hier etwas generischer, aber im Prinzip wie oben:
	
	`PERFORM LESEN-DATENSATZ`
	
	Und die SECTION selbst:
	
	`LESEN-DATENSATZ SECTION.`
	`.`

Verarbeite, solange Sätze in der csv-Datei sind oder ein Fehler aufgetreten ist	Schleifen werden auch mit dem Schlüsselwort PERFORM eingeleitet, allerdings gefolgt von einem UNTIL. Abgeschlossen wird die Schleife mit END-PERFORM ohne Punkt. Alles was dazwischen liegt, wird solange wieder und wieder ausgeführt, bis die Abbruchbedingung erfüllt ist.
	
	Die beiden Zustände „Ende der Datei erreicht, ja oder nein?“ und „es ist ein Fehler aufgetreten ja oder nein“ würde man in Java als boolean deklarieren, das haben wir in COBOL nicht zur Verfügung. Wohl aber können wir uns selber über die 88er-Stufen ein „entweder/oder“ definieren. Die Deklarationerfolgt in der WORKING-STORAGE SECTION. In unserem Fall brauchen wir also 2 Schalter:
	
	`01 EINGABE-ENDE-ERREICHT  PIC X(1).`
	`     88 EINGABE-ENDE-JA      VALUE "J".`
	`     88 EINGABE-ENDE-NEIN    VALUE "N".`
	`01 FEHLER-SCHALTER        PIC X(1).`
	`     88 FEHLER-JA            VALUE "J".`
	`     88 FEHLER-NEIN          VALUE "N".`
	
	Wir haben hier die Möglichkeit noch weitere 88er Stufen einbauen, also beispielsweise `FEHLER-VIELLEICHT VALUE "V".`, was natürlich nicht sehr sinnvoll ist, wir haben damit aber mehr Möglichkeiten als im boolean-Format.
	
	Damit sieht unsere Schleife so aus:
	
	`PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA`
	`  <Anweisungen>`

	END-PERFORM Da wir den ersten Datensatz schon gelesen haben, wissen wir, ob die Datei leer ist oder bis hierhin schon ein Verarbeitungsfehler aufgetreten ist. Falls das der Fall ist, würden wir gar nicht erst in die Verarbeitung springen. Letzte Aktivität in unserer Schleife muss dann wieder der Aufruf von LESEN-DATENSATZ sein. Die beiden nächsten Punkte in der obigen Liste sind mit der Schleife implizit abgedeckt. „Prüfe, ob das Ende der Datei erreicht oder ein Fehler aufgetreten ist“ und „“ ist die Anweisung *nach* dem UNTIL, und *das UNTIL selbst* ist die Aussage „Wenn eins von beiden zutrifft, dann beende die Verarbeitung“.
Sonst gehe in die Konvertierung	Das ist jetzt wieder keine große Kunst, das können wir schon: PERFORM KONVERTIERE-DATENSATZ Und die SECTION selbst: KONVERTIERE-DATENSATZ SECTION. .
Prüfe dort zuerst, ob es der erste Datensatz ist	Analog der Abbruchbedingung für die Schleife, können wir uns auch hier einen Schalter definieren. 01 ERSTER-SATZ PIC X(1). 88 ERSTER-SATZ-JA VALUE "J". 88 ERSTER-SATZ-NEIN VALUE "N". In der INITIALISIERUNG SECTION setzen wir den Schalter auf ERSTER-SATZ-JA. Das machen wir mittels SET ERSTER-SATZ-JA TO TRUE Damit haben wir sichergestellt, dass wir zum Zeitpunkt der Abfrage den ersten Satz gelesen haben. Die Prüfung selbst geht mit IF Anweisung für den Wenn-Fall (ELSE Anweisung für den Sonst-Fall) END-IF Da wir innerhalb der Schleife immer wieder an diesem Punkt vorbeikommen, müssen wir nach dem ersten Gebrauch die Anweisung SET ERSTER-SATZ-NEIN TO TRUE abarbeiten lassen. Die Prüfung ist dann in unserm Fall IF ERSTER-SATZ-JA SET ERSTER-SATZ-NEIN-TO TRUE END-IF Und natürlich weiter unten noch die SECTION

	SCHRIEBEN-DATENSATZ SECTION. .
Wenn ja, schreibe Zeile 1 und Zeile 2 in die xml-Datei	Da bauen wir uns erst einmal nur die Hülle zum Schreiben eines Datensatzes, den konkreten Inhalt bekommen wir später. Dazu erweitern wir die Prüfung um den Aufruf einer neuen SECTION. IF ERSTER-SATZ-JA PERFORM SCHRIEBEN-DATENSATZ SET ERSTER-SATZ-NEIN-TO TRUE END-IF Und natürlich weiter unten noch die SECTION selbst SCHRIEBEN-DATENSATZ SECTION. .
Wenn nein, dann konvertiere Objekt für Objekt von csv nach xml und schreibe Zeile für Zeile in xml-Datei	Das wird ein größerer Happen, den führen wir uns hier noch nicht zu Gemüte. Für jetzt reicht uns, dass wir in die Konvertierung navigieren können und sobald ein Satz fertig umgewandelt ist, schreiben wir ihn in die xml-Datei. Den Aufruf haben wir ja schon mit „Sonst gehe in die Konvertierung“ implementiert.
Führe Abschlussarbeiten durch	Auch hier definieren wir: PERFORM ABSCHLUSSARBEITEN Und wieder die SECTION selbst: ABSCHLUSSARBEITEN SECTION. .
Schließe die beiden Dateien	Gleiches Muster: PERFORM SCHLIESSEN-DATEIEN Und auch hier die SECTION selbst: SCHLIESSEN-DATEIEN SECTION. .
Ende STEUER-SECTION und	Der letzte Befehl in der STEUER SECTION ist STOP RUN.
END PROGRAM YOUR-PROGRAM-NAME.	END PROGRAM YOUR-PROGRAM-NAME.

Damit sind wir schon soweit, alles in den Editor einzugeben.

Zwei aufeinander folgende Punkte in den SECTIONs findet der Compiler nicht gut, eine SECTION ohne konkrete Anweisung ist für ihn ein Fehler. Um den Compiler zu beruhigen, aber auch damit wir wissen, dass die SECTION-Aufrufe auch tatsächlich geklappt haben, bauen wir überall noch ein DISPLAY ein und lassen uns jeweils den SECTION-Namen ausgeben. Für die erste SECTION INITIALISIEREN sieht das dann so aus:

```
INITIALISIERUNG SECTION.
   DISPLAY „IN INITIALISIERUNG“
.
```

Wenn wir das Programm jetzt starten würden, hätten wir eine Endlosschleife, denn keine unserer Abbruchbedingungen „Ende der Datei“ oder „Fehler“ wird je erreicht. Um das zu ändern bauen wir noch in die LESEN-DATENSATZ SECTION eine IF-Abfrage ein.

Im 1. Durchlauf sollte der Schalter ERSTER-SATZ-JA gesetzt sein, das haben wir ja in der INITIALISIERUNG festgelegt. Vor dem 2. Durchlauf setzen wir ihn in der PERFORM-Schleife auf ERSTER-SATZ-NEIN.

Den Schalter fragen wir jetzt in der LESEN-DATENSATZ SECTION ab. Wenn er auf ERSTER-SATZ-NEIN steht, wissen wir, dass wir im 2. Durchgang sind und da setzen wir jetzt die eine Abbruchbedingung unserer Schleife, nämlich EINGABE-ENDE-JA. Die SECTION sieht dann so aus:

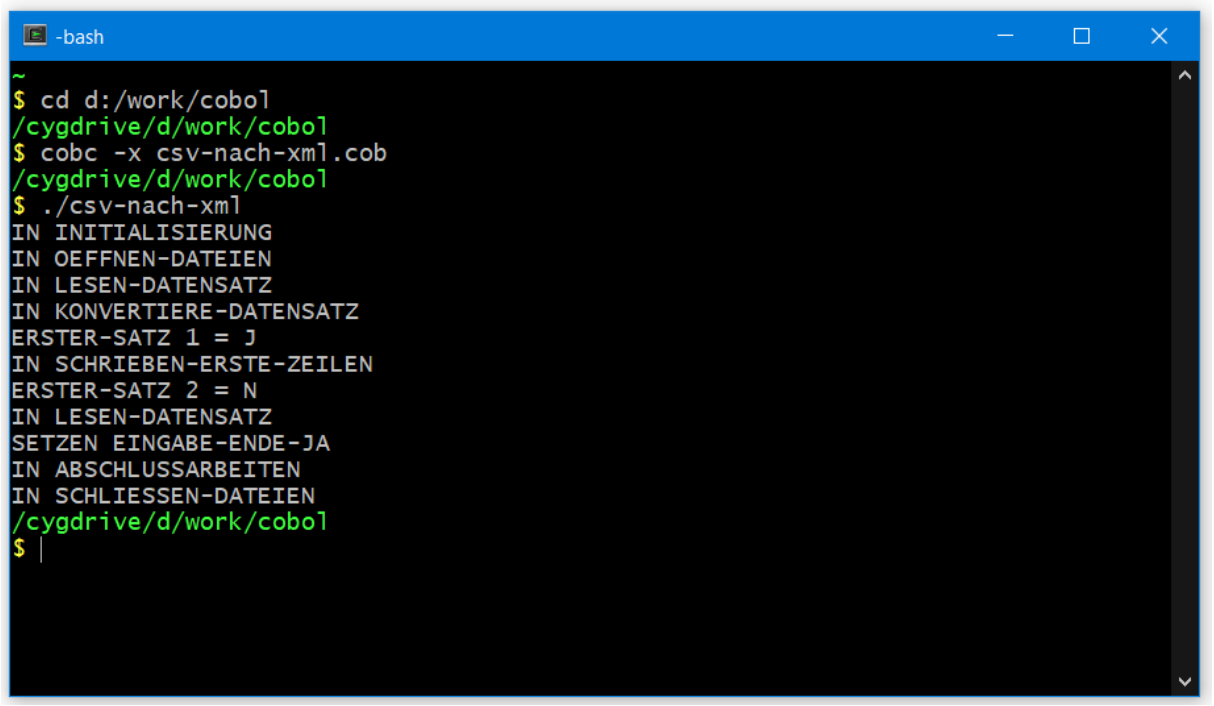
```
LESEN-DATENSATZ SECTION.
    DISPLAY "IN LESEN-DATENSATZ"
    IF ERSTER-SATZ-NEIN
       DISPLAY "SETZEN EINGABE-ENDE-JA"
       SET EINGABE-ENDE-JA TO TRUE
    END-IF
.
```

Bei mir sieht der Quellcode nach der Eingabe der Anweisungen so aus:

```
      ******************************************************************
      * Author:    papa
      * Date:      Juni 2021
      * Purpose:   Konvertieren von Inhalten aus csv nach xml
      *            Ausbaustufe 4.1 Die Programmstruktur
      * Tectonics: cobc
      ******************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CSV-NACH-XML.
       DATA DIVISION.
       FILE SECTION.
       WORKING-STORAGE SECTION.

      *Definition der Schalter
       01 EINGABE-ENDE-ERREICHT  PIC X(1).
           88 EINGABE-ENDE-JA        VALUE "J".
           88 EINGABE-ENDE-NEIN      VALUE "N".

       01 FEHLER-SCHALTER        PIC X(1).
           88 FEHLER-JA              VALUE "J".
           88 FEHLER-NEIN            VALUE "N".

       01 ERSTER-SATZ            PIC X(1).
           88 ERSTER-SATZ-JA         VALUE "J".
```

```
25            88 ERSTER-SATZ-NEIN       VALUE "N".
26
27         PROCEDURE DIVISION.
28
29        *Beginn der Steuerung
30         STEUER SECTION.
31             PERFORM INITIALISIERUNG
32             PERFORM OEFFNEN-DATEIEN
33             PERFORM LESEN-DATENSATZ
34             PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA
35                 PERFORM KONVERTIERE-DATENSATZ
36                 PERFORM LESEN-DATENSATZ
37             END-PERFORM
38             PERFORM ABSCHLUSSARBEITEN
39             PERFORM SCHLIESSEN-DATEIEN
40             STOP RUN.
41        *Ende der Steuerung
42
43        *Beginn der Prozeduren
44         INITIALISIERUNG SECTION.
45             DISPLAY "IN INITIALISIERUNG"
46        *    Setzen der Schalter auf Anfangszustand
47             SET EINGABE-ENDE-NEIN TO TRUE
48             SET FEHLER-NEIN TO TRUE
49             SET ERSTER-SATZ-JA TO TRUE
50         .
51         OEFFNEN-DATEIEN SECTION.
52             DISPLAY "IN OEFFNEN-DATEIEN"
53         .
54         LESEN-DATENSATZ SECTION.
55             DISPLAY "IN LESEN-DATENSATZ"
56             IF ERSTER-SATZ-NEIN
57                DISPLAY "SETZEN EINGABE-ENDE-JA"
58                SET EINGABE-ENDE-JA TO TRUE
59             END-IF
60         .
61         KONVERTIERE-DATENSATZ SECTION.
62             DISPLAY "IN KONVERTIERE-DATENSATZ"
63
64             DISPLAY "ERSTER-SATZ 1 = " ERSTER-SATZ
65             IF ERSTER-SATZ-JA
66                PERFORM SCHRIEBEN-DATENSATZ
67             END-IF
68             SET ERSTER-SATZ-NEIN TO TRUE
69             DISPLAY "ERSTER-SATZ 2 = " ERSTER-SATZ
70         .
71         SCHRIEBEN-DATENSATZ SECTION.
72             DISPLAY "IN SCHRIEBEN-DATENSATZ"
73         .
74         ABSCHLUSSARBEITEN SECTION.
75             DISPLAY "IN ABSCHLUSSARBEITEN"
76         .
77         SCHLIESSEN-DATEIEN SECTION.
78             DISPLAY "IN SCHLIESSEN-DATEIEN"
79         .
80         END PROGRAM CSV-NACH-XML.
81
```

Sieht bei Euch auch so aus? Na, dann wollen wir es testen. Also Quellcode speichern, Cygwin-Terminal aufmachen, in den Ordner des Codes navigieren, mit „cobc“ die .exe erstellen und dann mit ./ laufen lassen.



Sieht prima aus, durch die DISPLAY-Anweisungen wissen wir, wo das Programm durchgelaufen ist.

Im nächsten Kapitel kümmern wir uns um die Verarbeitung der Eingabe.

4.2. Lesen der csv-Datei

Zur Vorbereitung auf das Lesen in unserem Programm sollten wir uns zuerst eine Kopie der „kennzeichen.csv“ anlegen, die nur die ersten 3 Datensätze enthält. Dann braucht das Programm beim Durchlaufen nicht alle 700 Einträge durchzugehen. Meine Kopie heißt kennzeichen-klein.csv.

Im Programm müssen wir an mehreren Stellen Hand anlegen. Fangen wir von oben an.

4.2.1. Erweiterung ENVIRONMENT DIVISION

Die ENVIRONMENT DIVISION ist der Ort, an dem sie Umgebungsparameter des Programms zusammengetragen werden. Ihr Platz ist vor der DATA DIVISION.

Für die Behandlung von Ein- oder Ausgabedateien ist die INPUT-OUTPUT SECTION verantwortlich, hier gibt es die Abteilung FILE-CONTROL.

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

SELECT EINGABE ASSIGN TO 'kennzeichen-klein.csv'
ORGANIZATION IS LINE SEQUENTIAL
FILE STATUS IS EINGABE-STATUS.
```

Die Zeile `ORGANIZATION IS LINE SEQUENTIAL` gibt an, dass die Daten in der Datei so organisiert sind, dass jedes Zeilenende mit einem Zeilenumbruch gekennzeichnet ist.

Den `FILE STATUS` brauchen wir für das Erkennen des Endes der Datei, den definieren wir in der FILE-SECTION, die findet sich in der DATA DIVISION. Wenn alles in Ordnung ist, ist der Status 00. Gibt es keinen weiteren Datensatz mehr, ist der Status 10. Alle anderen Status sind Fehler.

4.2.2. Erweiterung DATA DIVISION

Hier haben wir mehrere Erweiterungen zu machen, die erste in der FILE SECTION. Hier müssen wir die Satz-Länge der Eingabedatei angeben. Dazu müssen wir uns den längsten Datensatz aus der Eingabe suchen. Es gibt 2 Datensätze, die je 122 Zeichen lang sind (Zeilennummern 408 und 417). Damit wir auf der sicheren Seite sind, legen wir eine Länge von 130 Zeichen fest.

```
FD  EINGABE.
01  EINGABE-FILE.
    05 EINGABE-GANZ       PIC X(130).
```

Wir haben das `PIC X(irgendwas)` jetzt schon öfter benutzt, ich denke hier ist ein guter Ort das zu erklären. „PIC“ steht für „PICTURE“ oder auch „PICTURE CLAUSE“ und soll den Inhalt des Feldes spezifizieren. Die Zahl in der Klammer gibt die Länge an.

In unserem Fall steht das X für „sowohl numerische Inhalte erlaubt, als auch Buchstaben“. Beinhaltet ein Feld nur numerische Werte, also nur Zahlen, wäre die PIC-Angabe `PIC 9(irgendwas)`. Mit PIC-9-Feldern kann man auch rechnen. Reine Buchstaben-Werte in Feldern wäre PIC A, in der Praxis hat sich aber durchgesetzt das X zu verwenden.

Der Grund ist relativ einfach zu erklären, schleicht sich eine Zahl in ein PIC-A-Feld gibt es einen Programmfehler. Mit PIC X ist man also auch damit auf der sicheren Seite.

Fließkommazahlen gibt es auch, da ist dann ein V mit verbaut, das ist die Stelle, an der ein Komma gesetzt wird. Naja, eigentlich ein Punkt, da in der amerikanischen Schreibweise von Zahlen die Tausendertrennung ein Komma ist und der Punkt für die Trennung der Nachkomma-Stellen steht. Als Beispiel hier PIC 9(5)V9(3) sind 5 Vorkomma-Stellen und 3 Nachkomma-Stellen.

Da wir hier nicht rechnen müssen, noch ein Hinweis. COBOL ist da sehr spaßbefreit, ein 3-stellig numerisches Feld kann niemals den Wert „1000“ haben. Machen wir ein Beispiel.

Wir haben ein Feld SUMME PIC 9(03) und weisen ihm den Wert 999 zu. Das geht mit MOVE 999 TO SUMME. Wenn wir jetzt eine 1 addieren, würden wir bei 1000 landen. Nicht COBOL, das füllt rechtsbündig auf und damit hat das Feld SUMME nach der Addition den Wert 000, da nur die letzten 3 Zahlen enthalten sind. Man nennt das dann „Überlauf“.

In der WORKING-STORAGE SECTION müssen wir wie oben beschrieben noch den `EINGABE-STATUS` definieren, das Statusfeld ist ein 2-stelliger Code aus Zahlen, wir geben also an

```
05 EINGABE-STATUS        PIC 9(2).
```

Daneben brauchen wir noch einen Ort, an dem wir den eingelesenen Datensatz speichern können. COBOL erwartet, dass die Eingabe nicht editiert wird, wir wollen aber ja mit den Inhalten etwas anstellen, von daher brauchen wir eine Variable in der WORKING-STORAGE SECTION in die wir den Datensatz ablegen können. Bei uns ist das

```
05 WS-EINGABE            PIC X(130).
```

Die Nummerierung vor dem Variablennamen zeigt die Hierarchie an. Auf der Stufe oberhalb der 05 gibt es noch die 01, das sind alle unsere Variablen. Statt einer 05 hätten wir auch eine 02 machen können, die 5er Schritte werden zur Sicherheit gewählt. Falls eine Gruppierung von 05er Stufen erfolgen soll, muss nicht alles überarbeitet werden, sondern nur eine 02er Stufe dazu genommen werden.

```
01  VARIABLEN.
    05 EINGABE-STATUS       PIC 9(2).
    05 WS-EINGABE           PIC X(130).
```

Damit bei Programmstart alle Variablen auch tatsächlich leer oder 0 sind, müssen wir sie initialisieren. Das machen wir in der INITIALISIERUNG SECTION mit

```
INITIALIZE VARIABLEN
```

Das sorgt dafür, dass PIC-X-Felder mit Spaces und PIC-9-Felder mit Nullen belegt werden.

Für die Abfrage des FILE-STATUS brauchen wir noch eine Konstante, nämlich die 10. Der Status 00 sind ZEROES, die können wir direkt abfragen. Wir definieren also noch

```
01  KONSTANTEN.
    05 NUM-10               PIC 9(2) VALUE 10.
```

Konstanten sind in der Deklaration genau wie Variablen, allerdings zeigt ein VALUE und ein Wert an, mit was das Feld bestückt wird. In unserem Fall mit einer 10.

Damit sind wir erstmal fertig in der DATA DIVISION.

4.2.3. Erweiterung PROCEDURE DIVISION

Auch hier müssen wir mehrere Dinge tun. Die INITIALISIEREN SECTION erweitern wir um

```
*     Setzen der Schalter auf Anfangszustand
      SET EINGABE-ENDE-NEIN TO TRUE
      SET FEHLER-NEIN TO TRUE
      SET ERSTER-SATZ-JA TO TRUE

*     Alle Variablen auf einen Schlag initialisieren
      INITIALIZE VARIABLEN
```

Bevor wir die Datei lesen können, müssen wir sie öffnen. Dafür haben wir ja die OEFFNEN-DATEIEN SECTION. Hier ergänzen wir

```
OPEN INPUT EINGABE
```

Und wo ein OPEN ist, ist auch ein CLOSE, wir haben die SCHLIESSEN-DATEIEN SECTION, da fügen wir folgendes ein:

```
CLOSE EINGABE
```

In der LESEN-DATENSATZ SECTION schmeißen wir die IF-Abfrage raus, dafür nehmen wir jetzt den READ-Befehl mit auf:

```
READ EINGABE INTO WS-EINGABE
   AT END SET EINGABE-ENDE-JA TO TRUE
   NOT AT END DISPLAY EINGABE-GANZ
END-READ
```

Der Code erklärt sich von selbst, oder? Die Anweisung ist, „Lies den aktuellen Datensatz der EINGABE in die WS-EINGABE, wenn das Ende erreicht ist, setze den Schalter `EINGABE-ENDE-JA`, sonst zeige EINGABE-GANZ auf der Konsole an.“

Das ist das Schöne an COBOL, es ist so nah an der gesprochenen Sprache, dass vieles selbsterklärend ist.

Da wir gute Entwickler sind, fragen wir hier auch noch den Status des Leseversuchs ab. Wir erinnern uns, alles okay ist 00 und 10 ist Ende der Datei. Alles andere ist ein Fehler.

```
EVALUATE EINGABE-STATUS
   WHEN ZEROES
   WHEN NUM-10
      CONTINUE
   WHEN OTHER
      DISPLAY "EINGABE-STATUS = " EINGABE-STATUS
      SET FEHLER-JA TO TRUE
END-EVALUATE
```

ZERO und ZEROES sind reservierte Worte und somit schon bekannt, die 10 haben wir ja schon als Konstante definiert.

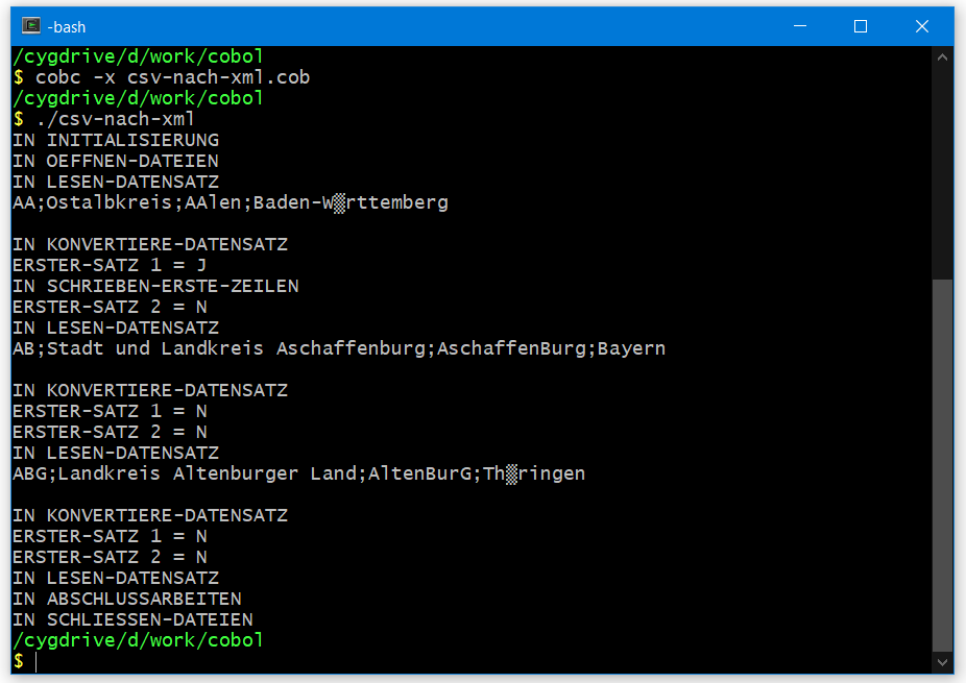
Wenn Ihr auch das nach und nach eingebaut habt, sollte der Code jetzt so aussehen:

```
1       ******************************************************************
2       * Author:    papa
3       * Date:      Juni 2021
4       * Purpose:   Konvertieren von Inhalten aus csv nach xml
5       *            Ausbaustufe 4.2 Lesen der csv-Datei
6       * Tectonics: cobc
7       ******************************************************************
8        IDENTIFICATION DIVISION.
9        PROGRAM-ID. CSV-NACH-XML.
10       ENVIRONMENT DIVISION.
11       INPUT-OUTPUT SECTION.
12       FILE-CONTROL.
13
14       SELECT EINGABE ASSIGN TO 'kennzeichen-klein.csv'
15       ORGANIZATION IS LINE SEQUENTIAL
16       FILE STATUS IS EINGABE-STATUS.
17
18       DATA DIVISION.
19       FILE SECTION.
20
21       FD EINGABE.
22       01 EINGABE-FILE.
23           05 EINGABE-GANZ            PIC X(130).
24
25       WORKING-STORAGE SECTION.
26
27      *Definition der Schalter
28       01 EINGABE-ENDE-ERREICHT  PIC X(1).
29           88 EINGABE-ENDE-JA        VALUE "J".
30           88 EINGABE-ENDE-NEIN      VALUE "N".
31
32       01 FEHLER-SCHALTER        PIC X(1).
33           88 FEHLER-JA              VALUE "J".
34           88 FEHLER-NEIN            VALUE "N".
35
36       01 ERSTER-SATZ            PIC X(1).
37           88 ERSTER-SATZ-JA         VALUE "J".
38           88 ERSTER-SATZ-NEIN       VALUE "N".
39
40      *Definition der Variablen
41       01 VARIABLEN.
42           05 EINGABE-STATUS        PIC 9(2).
43           05 WS-EINGABE            PIC X(130).
44
45      *Definition der Konstanten
46       01 KONSTANTEN.
47           05 NUM-10                PIC 9(2) VALUE 10.
48       PROCEDURE DIVISION.
49
50      *Beginn der Steuerung
51       STEUER SECTION.
52           PERFORM INITIALISIERUNG
53           PERFORM OEFFNEN-DATEIEN
```

```
54            PERFORM LESEN-DATENSATZ
55            PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA
56                PERFORM KONVERTIERE-DATENSATZ
57                PERFORM LESEN-DATENSATZ
58            END-PERFORM
59            PERFORM ABSCHLUSSARBEITEN
60            PERFORM SCHLIESSEN-DATEIEN
61            STOP RUN.
62       *Ende der Steuerung
63
64       *Beginn der Prozeduren
65        INITIALISIERUNG SECTION.
66            DISPLAY "IN INITIALISIERUNG"
67
68       *    Setzen der Schalter auf Anfangszustand
69            SET EINGABE-ENDE-NEIN TO TRUE
70            SET FEHLER-NEIN TO TRUE
71            SET ERSTER-SATZ-JA TO TRUE
72
73       *    Alle Variablen auf einen Schlag initialisieren
74            INITIALIZE VARIABLEN
75        .
76        OEFFNEN-DATEIEN SECTION.
77            DISPLAY "IN OEFFNEN-DATEIEN"
78            OPEN INPUT EINGABE
79        .
80        LESEN-DATENSATZ SECTION.
81            DISPLAY "IN LESEN-DATENSATZ"
82
83            READ EINGABE INTO WS-EINGABE
84               AT END SET EINGABE-ENDE-JA TO TRUE
85               NOT AT END DISPLAY EINGABE-GANZ
86            END-READ
87
88            EVALUATE EINGABE-STATUS
89               WHEN ZEROES
90               WHEN NUM-10
91                  CONTINUE
92               WHEN OTHER
93                  DISPLAY "EINGABE-STATUS = " EINGABE-STATUS
94                  SET FEHLER-JA TO TRUE
95            END-EVALUATE
96        .
97        KONVERTIERE-DATENSATZ SECTION.
98            DISPLAY "IN KONVERTIERE-DATENSATZ"
99
100           DISPLAY "ERSTER-SATZ 1 = " ERSTER-SATZ
101           IF ERSTER-SATZ-JA
102              PERFORM SCHRIEBEN-DATENSATZ
103           END-IF
104           SET ERSTER-SATZ-NEIN TO TRUE
105           DISPLAY "ERSTER-SATZ 2 = " ERSTER-SATZ
106       .
107       SCHRIEBEN-DATENSATZ SECTION.
108           DISPLAY "IN SCHRIEBEN-DATENSATZ"
109       .
110       ABSCHLUSSARBEITEN SECTION.
```

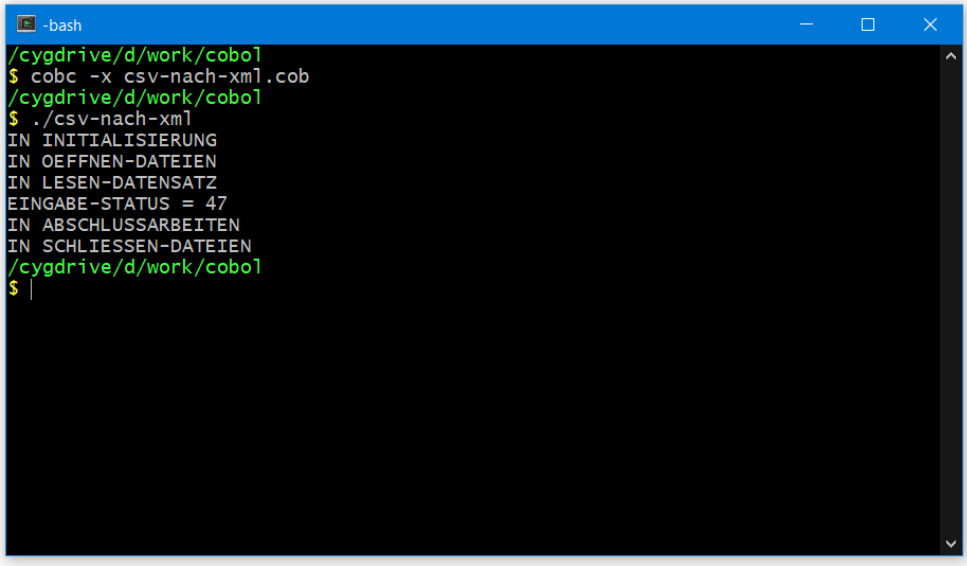
```
111             DISPLAY "IN ABSCHLUSSARBEITEN"
112         .
113         SCHLIESSEN-DATEIEN SECTION.
114             DISPLAY "IN SCHLIESSEN-DATEIEN"
115             CLOSE EINGABE
116         .
117         END PROGRAM CSV-NACH-XML.
118
```

Das sollten wir jetzt mal testen. Schauen wir uns das auch noch im Cygwin Terminal an:



Das sieht doch schon ganz gut aus. Um die falsche Darstellung des „ü“ kümmern wir uns später.

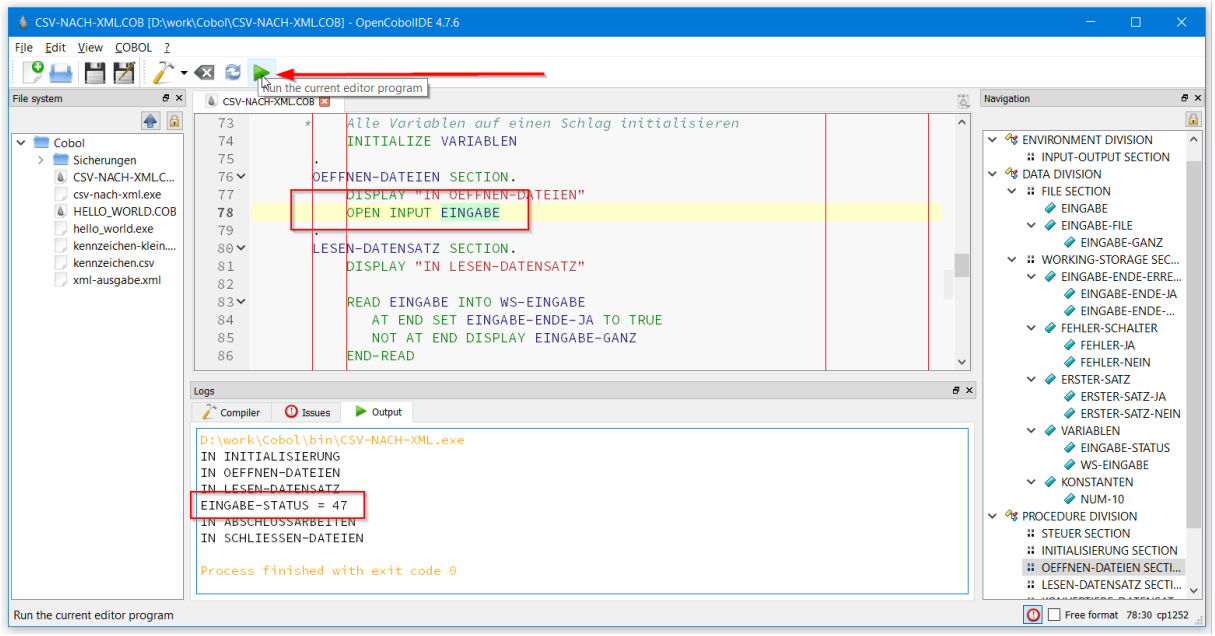
Wenn Ihr einen Dateiverarbeitungs-Fehler provozieren wollt, kommentiert die Zeile 78 (`OPEN INPUT EINGABE`) mal aus oder löscht sie. Dann sollte die Ausgabe so aussehen:



Beim Zugriff auf die Datei gibt es den Status-Code 47, was eben bedeutet, dass wir versucht haben auf eine Ressource zuzugreifen, die für das Programm noch nicht zugänglich ist.

Offensichtlich wurde aber der Fehler-Schalter gesetzt, denn die Verarbeitung ist nicht angelaufen. Also schnell wieder rein mit dem Statement.

Und noch ein Hinweis, ab hier können wir den Run-Pfeil in der IDE nicht mehr nutzen. Auch wenn die Zeile 78 korrekt eingebaut ist, gibt es beim Lauf mit Klick auf den grünen Pfeil die Statusmeldung 47:



Also, ab jetzt nur noch Cygwin!

Tipp – bindet mal die große Datei ein, das sollte genauso klappen.

4.3. Schreiben der xml-Datei

Als nächstes gehen wir das Schreiben der xml-Datei an. Wir haben ja schon gesehen, dass wir vor den ersten Daten 2 Zeilen zum Auftakt ausgeben müssen. Das sind

```
1 <?xml version="1.0"?>
2 <kennzeichen>
```

Dies werden wir in diesem Kapitel umsetzen.

Es ist nicht sehr verwunderlich, dass wir an den gleichen Stellen eingreifen müssen. Daher sind die weiteren Kapitel auch die gleichen wie in Kapitel 4.2.

4.3.1. Erweiterung ENVIRONMENT DIVISION

Hinter der Eingabe definieren wir die Ausgabe analog:

```
SELECT AUSGABE ASSIGN TO 'xml-ausgabe.xml'
ORGANIZATION IS LINE SEQUENTIAL
FILE STATUS IS AUSGABE-STATUS.
```

Den `FILE STATUS` kennen wir schon, den müssen wir in der DATA DIVISION noch definieren.

4.3.2. Erweiterung DATA DIVISION

Analog zur EINGABE müssen wir auch die AUSGABE definieren.

```
FD  AUSGABE.
01  AUSGABE-FILE.
    05 AUSGABE-ZEILE          PIC X(130).
```

In der WORKING-STORAGE SECTION müssen wir wie oben beschrieben noch den `AUSGABE-STATUS` definieren

```
    05 AUSGABE-STATUS         PIC 9(2).
```

Die ersten beiden Zeilen in der xml-Datei sind Konstanten, hier gibt es keinen variablen Anteil. Daher packen wir das auch in den Teil der KONSTANTEN.

Wir haben gelernt, dass die Werte der Konstanten in Anführungsstrichen stehen. Würden wir die erste Zeile komplett als Konstante eingeben, meckert der Compiler. Grund sind die beiden Anführungsstriche vor und nach der 1.0 (unten rot markiert). Das erste Anführungszeichen vor der 1 führt dazu, dass der Compiler meint, der VALUE-Wert ist zu Ende. Dann würde ein Punkt folgen müssen, der fehlt dem Compiler:

```
05 WS-AUSGABE.
   10 AUS-VERSION.
      15 FILLER PIC X(21) VALUE "<?xml version="1.0"?>".
```

Daher müssen wir jetzt ein wenig tricksen. Wir müssen den String in mehrere Teile teilen und dann noch ein Anführungszeichen in ein Anführungszeichen zu bringen.

Ich nehme es vorweg, der Code dafür ist folgender:

```
05 WS-AUSGABE.
   10 AUS-VERSION.
      15 FILLER          PIC X(14) VALUE "<?xml version=".
      15 FILLER          PIC X(01) VALUE X'22'.
      15 FILLER          PIC X(03) VALUE "1.0".
      15 FILLER          PIC X(01) VALUE X'22'.
      15 FILLER          PIC X(03) VALUE " ?>".
```

Hier sehen wir etwas, das wir noch nicht kennen. Die Konstante als Ganzes sprechen wir über den Namen AUS-VERSION an. Die einzelnen Teile brauchen wir nicht separat, daher sind sie hier als FILLER deklariert.

Der zweite und der vierte FILLER haben noch eine Besonderheit, hinter dem VALUE steht ein X und dann die einfachen Anführungsstriche. Das bedeutet, dass der Wert „22“ in der hexadezimalen Darstellung ist. Und hexadezimal „22“ ist das doppelte Anführungszeichen „“.

Die zweite Zeile ist dann wieder einfach, sie besteht aus der Zeichenkette `<kennzeichen>`. Das lässt sich in einer Zeile abbilden und sieht dann so aus

```
10 AUS-KENNZEICHEN-AUF PIC X(13) VALUE "<kennzeichen>".
```

Da wir wissen, dass die letzte Zeile `</kennzeichen>` ist, bauen wir die auch gleich mit ein:

```
10 AUS-KENNZEICHEN-ZU PIC X(14) VALUE "</kennzeichen>".
```

Wie wir unten lernen werden, erfolgt das Schreiben eines Datensatzes über die Anweisung `WRITE AUSGABE-FILE FROM „working storage variable“`. Daher müssen wir uns die Variable noch definieren. Ich nenne sie

```
01 VARIABLEN.
   …
   05 TEMP-AUSGABE-ZEILE        PIC X(130).
```

Damit sind wir erstmal fertig in der DATA DIVISION.

4.3.3. Erweiterung PROCEDURE DIVISION

Analog zur Eingabe - bevor wir die Datei schreiben können, müssen wir sie öffnen. Dafür haben wir ja die OEFFNEN-DATEIEN SECTION. Hier ergänzen wir

```
OPEN OUTPUT AUSGABE
```

Und auch hier in der SCHLIESSEN-DATEIEN SECTION

```
CLOSE AUSGABE
```

Für das Schreiben der Daten haben wir die SCHREIBEN-DATENSATZ SECTION eingerichtet. Die Anweisung für das Schreiben kennen wir schon, wir müssen sie nur noch einbauen. Und auch hier fragen wir den Status ab, wobei alles was größer als 00 ist als Fehler ausgegeben

```
SCHRIEBEN-DATENSATZ SECTION.
    DISPLAY "IN SCHRIEBEN-DATENSATZ"

    WRITE AUSGABE-FILE FROM TEMP-AUSGABE-ZEILE

    IF AUSGABE-STATUS > ZEROES
        DISPLAY "AUSGABE-STATUS = " AUSGABE-STATUS
        SET FEHLER-JA TO TRUE
    END-IF

    INITIALIZE TEMP-AUSGABE-ZEILE
.
```

Damit sieht unsere Verarbeitung immer gleich aus. Wann immer nach der Konvertierung eine neue Zeile in die xml-Datei eingefügt werden soll, sammeln wir alles zusammen, stecken es in die TEMP-AUSGABE-ZEILE und machen den Aufruf `PERFORM SCHRIEBEN-DATENSAATZ`.

Nachdem wir den Satz aus der temporären Variablen geschrieben haben, leeren wir sie wieder mit INITIALIZE, die Anweisung kennen wir aus der INITIALISIERUNG SECTION.

Wenden wir uns jetzt der KONVERTIEREN SECTION zu.

Die beiden DISPLAYs fliegen raus, das Funktionieren haben wir ja erfolgreich geprüft.

Die IF-Abfrage bleibt aber drin. Hier kopieren wir jetzt zuerst `AUS-VERSION` in die `TEMP-AUSGABE-ZEILE`, dann erfolgt das Schreiben mittels Aufruf SCHREIBEN-DATENSATZ. Das gleiche noch einmal für `AUS-KENNZEICHEN-AUF`.

Die Schalteränderung `ERSTER-SATZ-NEIN` kommt noch mit in die IF-Anweisung. Damit sieht die KONVERTIERE-DATENSATZ SECTION jetzt so aus:

```
 KONVERTIERE-DATENSATZ SECTION.
    DISPLAY "IN KONVERTIERE-DATENSATZ"

    IF ERSTER-SATZ-JA
       MOVE AUS-VERSION            TO TEMP-AUSGABE-ZEILE
       PERFORM SCHRIEBEN-DATENSATZ
       MOVE AUS-KENNZEICHEN-AUF    TO TEMP-AUSGABE-ZEILE
       PERFORM SCHRIEBEN-DATENSATZ
       SET ERSTER-SATZ-NEIN        TO TRUE
    END-IF

.
```

Der Inhalt von `AUS-KENNZEICHEN-ZU` ist das letzte, was in die xml-Datei geschrieben wird. Daher kommt das in die ABSCHLUSSARBEITEN:

```
ABSCHLUSSARBEITEN SECTION.
    DISPLAY "IN ABSCHLUSSARBEITEN"
    MOVE AUS-KENNZEICHEN-ZU         TO TEMP-AUSGABE-ZEILE
    PERFORM SCHRIEBEN-DATENSATZ
.
```

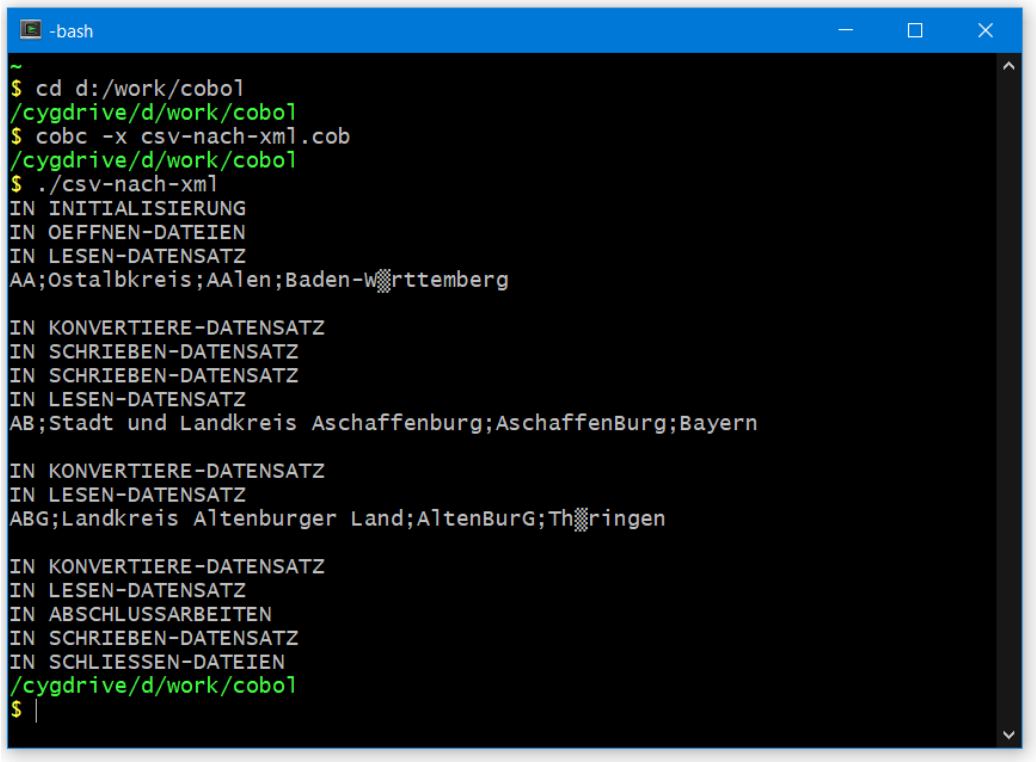
Das war es in der PROCEDURE DIVISION, der Quellcode sollte jetzt etwa so aussehen:

```
       ******************************************************************
       * Author:    papa
       * Date:      Juni 2021
       * Purpose:   Konvertieren von Inhalten aus csv nach xml
       *            Ausbaustufe 4.3. Schreiben der xml-Datei
       * Tectonics: cobc
       ******************************************************************
        IDENTIFICATION DIVISION.
        PROGRAM-ID. CSV-NACH-XML.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.

        SELECT EINGABE ASSIGN TO 'kennzeichen-klein.csv'
        ORGANIZATION IS LINE SEQUENTIAL
        FILE STATUS IS EINGABE-STATUS.

        SELECT AUSGABE ASSIGN TO 'xml-ausgabe.xml'
        ORGANIZATION IS LINE SEQUENTIAL
        FILE STATUS IS AUSGABE-STATUS.

        DATA DIVISION.
        FILE SECTION.

        FD EINGABE.
        01 EINGABE-FILE.
            05 EINGABE-GANZ             PIC X (130).

        FD AUSGABE.
        01 AUSGABE-FILE.
            05 AUSGABE-ZEILE            PIC X(130).

        WORKING-STORAGE SECTION.

       *Definition der Schalter
        01 EINGABE-ENDE-ERREICHT  PIC X(1).
            88 EINGABE-ENDE-JA        VALUE "J".
            88 EINGABE-ENDE-NEIN      VALUE "N".

        01 FEHLER-SCHALTER        PIC X(1).
            88 FEHLER-JA              VALUE "J".
            88 FEHLER-NEIN            VALUE "N".

        01 ERSTER-SATZ            PIC X(1).
            88 ERSTER-SATZ-JA         VALUE "J".
            88 ERSTER-SATZ-NEIN       VALUE "N".

       *Definition der Variablen
        01 VARIABLEN.
            05 EINGABE-STATUS        PIC 9(2).
            05 AUSGABE-STATUS        PIC 9(2).
            05 WS-EINGABE            PIC X(130).
            05 TEMP-AUSGABE-ZEILE    PIC X(130).

       *Definition der Konstanten
```

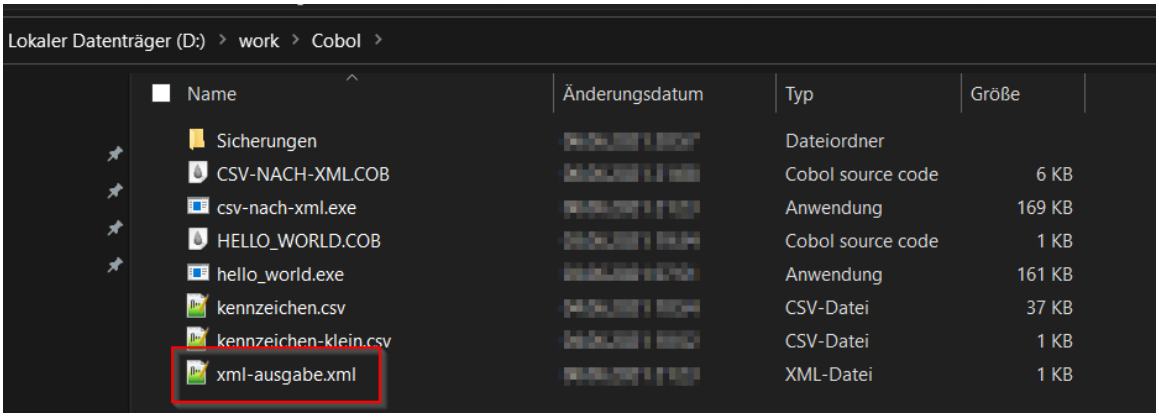
```
 56        01 KONSTANTEN.
 57            05 NUM-10                PIC 9(2) VALUE 10.
 58            05 WS-AUSGABE.
 59               10 AUS-VERSION.
 60                  15 FILLER           PIC X(14) VALUE "<?xml version=".
 61                  15 FILLER           PIC X(01) VALUE X'22'.
 62                  15 FILLER           PIC X(03) VALUE "1.0".
 63                  15 FILLER           PIC X(01) VALUE X'22'.
 64                  15 FILLER           PIC X(03) VALUE " ?>".
 65               10 AUS-KENNZEICHEN-AUF PIC X(13) VALUE "<kennzeichen>".
 66               10 AUS-KENNZEICHEN-ZU PIC X(14) VALUE "</kennzeichen>".
 67        PROCEDURE DIVISION.
 68
 69       *Beginn der Steuerung
 70        STEUER SECTION.
 71            PERFORM INITIALISIERUNG
 72            PERFORM OEFFNEN-DATEIEN
 73            PERFORM LESEN-DATENSATZ
 74            PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA
 75                PERFORM KONVERTIERE-DATENSATZ
 76                PERFORM LESEN-DATENSATZ
 77            END-PERFORM
 78            PERFORM ABSCHLUSSARBEITEN
 79            PERFORM SCHLIESSEN-DATEIEN
 80            STOP RUN.
 81       *Ende der Steuerung
 82
 83       *Beginn der Prozeduren
 84        INITIALISIERUNG SECTION.
 85            DISPLAY "IN INITIALISIERUNG"
 86       *    Setzen der Schalter auf Anfangszustand
 87            SET EINGABE-ENDE-NEIN TO TRUE
 88            SET FEHLER-NEIN TO TRUE
 89            SET ERSTER-SATZ-JA TO TRUE
 90
 91       *    Alle Variablen auf einen Schlag initialisieren
 92            INITIALIZE VARIABLEN
 93        .
 94        OEFFNEN-DATEIEN SECTION.
 95            DISPLAY "IN OEFFNEN-DATEIEN"
 96            OPEN INPUT EINGABE
 97            OPEN OUTPUT AUSGABE
 98        .
 99        LESEN-DATENSATZ SECTION.
100            DISPLAY "IN LESEN-DATENSATZ"
101
102            READ EINGABE INTO WS-EINGABE
103               AT END SET EINGABE-ENDE-JA TO TRUE
104               NOT AT END DISPLAY EINGABE-GANZ
105            END-READ
106
107            EVALUATE EINGABE-STATUS
108               WHEN ZEROES
109               WHEN NUM-10
110                  CONTINUE
111               WHEN OTHER
112                  DISPLAY "EINGABE-STATUS = " EINGABE-STATUS
```

```
113                 SET FEHLER-JA TO TRUE
114             END-EVALUATE
115         .
116         KONVERTIERE-DATENSATZ SECTION.
117             DISPLAY "IN KONVERTIERE-DATENSATZ"
118
119             IF ERSTER-SATZ-JA
120                 MOVE AUS-VERSION            TO TEMP-AUSGABE-ZEILE
121                 PERFORM SCHRIEBEN-DATENSATZ
122                 MOVE AUS-KENNZEICHEN-AUF    TO TEMP-AUSGABE-ZEILE
123                 PERFORM SCHRIEBEN-DATENSATZ
124                 SET ERSTER-SATZ-NEIN        TO TRUE
125             END-IF
126
127             MOVE AUS-KENNZEICHEN-ZU         TO TEMP-AUSGABE-ZEILE
128             PERFORM SCHRIEBEN-DATENSATZ
129         .
130         SCHRIEBEN-DATENSATZ SECTION.
131             DISPLAY "IN SCHRIEBEN-DATENSATZ"
132
133             WRITE AUSGABE-FILE FROM TEMP-AUSGABE-ZEILE
134
135             IF AUSGABE-STATUS > ZEROES
136                 DISPLAY "AUSGABE-STATUS = " AUSGABE-STATUS
137                 SET FEHLER-JA TO TRUE
138             END-IF
139
140             INITIALIZE TEMP-AUSGABE-ZEILE
141         .
142         ABSCHLUSSARBEITEN SECTION.
143             DISPLAY "IN ABSCHLUSSARBEITEN"
144             MOVE AUS-KENNZEICHEN-ZU         TO TEMP-AUSGABE-ZEILE
145             PERFORM SCHRIEBEN-DATENSATZ
146         .
147         SCHLIESSEN-DATEIEN SECTION.
148             DISPLAY "IN SCHLIESSEN-DATEIEN"
149             CLOSE EINGABE
150             CLOSE AUSGABE
151         .
152         END PROGRAM CSV-NACH-XML.
153             CLOSE AUSGABE
154         .
155         END PROGRAM CSV-NACH-XML.
156
```

Okay, testen wir das:

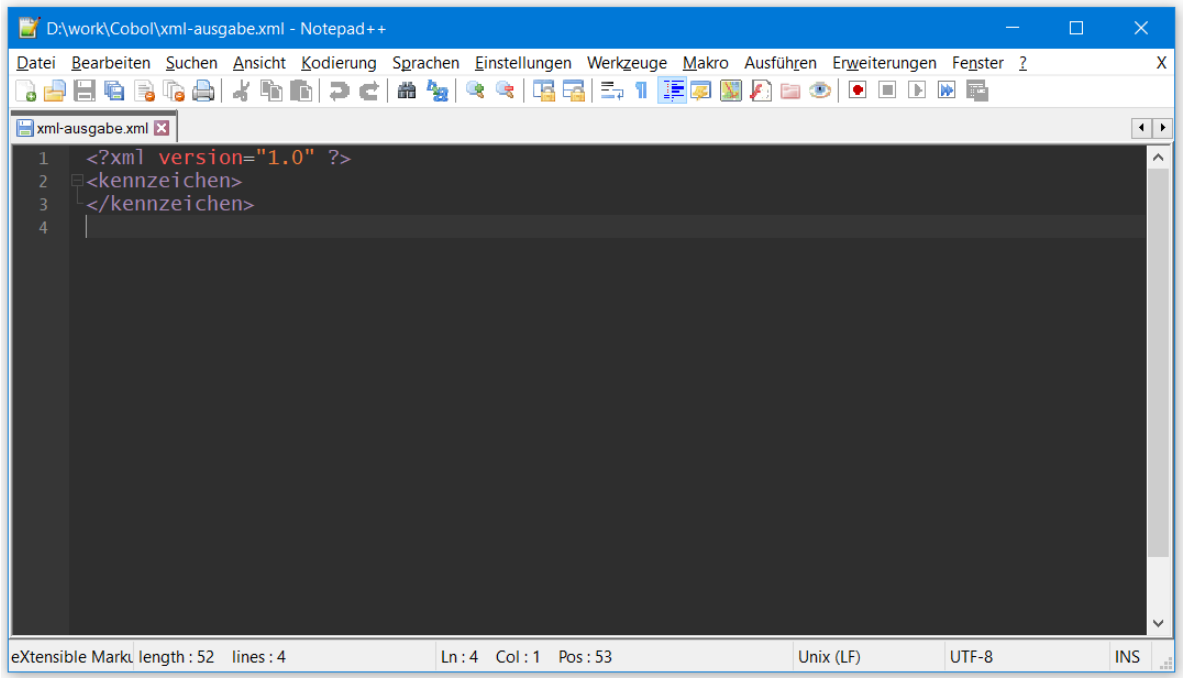


Die Ausgaben sehen gut aus, aber was ist mit der Ausgabe-Datei?



Da ist sie!

Und was ist der Inhalt?



Jap, alles wie es sein soll.

Ab hier haben wir das Grundgerüst fertig. Die nächsten Arbeiten finden in der `KONVERTIERE-DATENSATZ SECTION` statt.

4.4. Ausgabe der xml-tags

Wenden wir uns jetzt der Konvertierung der csv-Daten nach xml zu.

Das Aussehen der xml-Datei war folgendes:

```
<?xml version="1.0"?>
<kennzeichen>
    <record>
        <Abk>A</Abk>
        <Stadt_Landkreis>Stadt und Landkreis Augsburg</Stadt_Landkreis>
        <abgeleitet_von>Augsburg</abgeleitet_von>
        <Bundesland>Bayern</Bundesland>
    </record>
    <record>
        <Abk>AA</Abk>
        <Stadt_Landkreis>Ostalbkreis</Stadt_Landkreis>
        <abgeleitet_von>AAlen</abgeleitet_von>
        <Bundesland>Baden-Württemberg</Bundesland>
    </record>
</kennzeichen>
```

Je Datensatz in der csv-Datei müssen 6 Schreib-Vorgänge in die xml-Datei erfolgen.

Los geht es mit dem öffnenden `<record>`-tag, der 6. Schreib-Vorgang ist das schließende `</record>`-tag. Die 4 Zeilen dazwischen bestehen aus jeweils 3 Teilen, dem öffnenden tag, dem eigentlichen Wert und dem schließenden tag.

Um diese Aufgabe zu erfüllen, bedienen wir uns der beiden von COBOL bereitgestellten UNSTRING- und STRING-Anweisungen.

Mit UNSTRING werden Datensätze auf mehrere kleine Felder geteilt, mit STRING werden mehrere kleine Felder zu einem großen zusammengefügt.

Der generelle Aufbau der UNSTRING-Anweisung ist

```
UNSTRING FELD-GROSS DELIMITED BY [trennzeichen]
INTO FELD-1-KLEIN, <-- wichtig hier die Kommata je kleinem Feld!
     FELD-2-KLEIN,
     FELD-3-KLEIN
END-UNSTRING
```

Als Trennzeichen kann alles genutzt werden, in unserem Fall ist es das Semikolon. Wichtig hier, das Trennzeichen selbst wird nicht mitgenommen, es landet nicht in einem der kleinen Felder.

Der generelle Aufbau der STRING-Anweisung ist

```
STRING FELD-1 FELD-2
       FELD-3
INTO GROSSES-FELD
END-STRING
```

Das bauen wir jetzt beides in die KONVERTIEREN SECTION ein. Die ganze Zeile des csv-Satzes wurde in `WS-EINGABE` gespeichert. Mit der UNSTRING-Anweisung können wir jetzt ganz bequem die einzelnen Teile in unterschiedliche Felder schreiben, die Trennung erfolgt mittels Semikolons.

```
UNSTRING WS-EINGABE DELIMITED BY CHAR-SEMICOLON
   INTO WS-EINGABE-ABK,
        WS-EINGABE-STADT,
        WS-EINGABE-ABGEL,
        WS-EINGABE-BULAND
END-UNSTRING
```

Das Semikolon ist eine Konstante und wird so deklariert:

```
05 CHAR-SEMICOLON        PIC X(1) VALUE ";".
```

Die anderen Felder müssen natürlich noch als Variablen deklariert werden, aber wie ist ihre tatsächliche Länge? Gute Frage, hier ist leider keine einfache Antwort möglich.

In COBOL muss alles mit fester Länge deklariert werden. Der Compiler „reserviert“ tatsächlich den Speicherplatz für die Werte, die in den Picture-Anweisungen hinterlegt sind. Andere Sprachen sind da weiter, COBOL nicht.

Damit wir erst einmal Ergebnisse produzieren, machen wir es uns einfach und deklarieren folgendes:

```
05 WS-INHALT-ABK        PIC X(3).
05 WS-INHALT-STADT      PIC X(80).
05 WS-INHALT-ABGEL      PIC X(80).
05 WS-INHALT-BULAND     PIC X(80).
```

Das entspricht natürlich nicht der Realität, die Stadt „Hof“ hat natürlich keine 80 Zeichen, sondern nur 3. Aber das soll uns jetzt erst einmal nicht stören.

Um die einzelnen Zeilen zusammenzustellen, nutzen wir jetzt die STRING-Anweisung und geben den AUF-tag, das jeweilige WS-INHALT-Feld und das ZU-tag mit.

Das müssen wir für alle 4 Felder machen. Der Code in der KONVERTIEREN SECTION sieht dann so aus:

```
STRING AUS-ABK-AUF
       WS-INHALT-ABK
       AUS-ABK-ZU
INTO TEMP-AUSGABE-ZEILE
PERFORM SCHRIEBEN-DATENSATZ

STRING AUS-STADT-AUF
       WS-INHALT-STADT
       AUS-STADT-ZU
INTO TEMP-AUSGABE-ZEILE
PERFORM SCHRIEBEN-DATENSATZ

STRING AUS-ABGEL-AUF
       WS-INHALT-ABGEL
       AUS-ABGEL-ZU
INTO TEMP-AUSGABE-ZEILE
PERFORM SCHRIEBEN-DATENSATZ
```

```
           STRING AUS-BULAND-AUF
                  WS-INHALT-BULAND
                  AUS-BULAND-ZU
           INTO TEMP-AUSGABE-ZEILE
           PERFORM SCHRIEBEN-DATENSATZ

           MOVE AUS-RECORD_ZU              TO TEMP-AUSGABE-ZEILE
           PERFORM SCHRIEBEN-DATENSATZ
```

Analog zum Kennzeichen definieren wir uns also die Konstanten zu den AUF-tags und den ZU-tags:

```
             10 AUS-RECORD_AUF       PIC X(13) VALUE "    <record>".
             10 AUS-RECORD_ZU        PIC X(14) VALUE "    </record>".
             10 AUS-ABK-AUF          PIC X(13) VALUE "        <Abk>".
             10 AUS-ABK-ZU           PIC X(07) VALUE "</Abk>".
             10 AUS-STADT-AUF        PIC X(25) VALUE
                "        <Stadt_Landkreis>".
             10 AUS-STADT-ZU         PIC X(18) VALUE
                "</Stadt_Landkreis>".
             10 AUS-ABGEL-AUF        PIC X(24) VALUE
                "        <abgeleitet_von>".
             10 AUS-ABGEL-ZU         PIC X(17) VALUE
                "</abgeleitet_von>".
             10 AUS-BULAND-AUF        PIC X(16) VALUE
                "        <Buland>".
             10 AUS-BULAND-ZU         PIC X(17) VALUE
                "</Buland>".
```

Damit sollte der Quellcode unseres Programms jetzt so aussehen:

```
 1       ******************************************************************
 2       * Author:    papa
 3       * Date:      Juni 2021
 4       * Purpose:   Konvertieren von Inhalten aus csv nach xml
 5       *            Ausbaustufe 4.4. Ausgabe der xml-tags
 6       * Tectonics: cobc
 7       ******************************************************************
 8        IDENTIFICATION DIVISION.
 9        PROGRAM-ID. CSV-NACH-XML.
10        ENVIRONMENT DIVISION.
11        INPUT-OUTPUT SECTION.
12        FILE-CONTROL.
13
14        SELECT EINGABE ASSIGN TO 'kennzeichen-klein.csv'
15        ORGANIZATION IS LINE SEQUENTIAL
16        FILE STATUS IS EINGABE-STATUS.
17
18        SELECT AUSGABE ASSIGN TO 'xml-ausgabe.xml'
19        ORGANIZATION IS LINE SEQUENTIAL
20        FILE STATUS IS AUSGABE-STATUS.
21
22        DATA DIVISION.
23        FILE SECTION.
24
25        FD EINGABE.
26        01 EINGABE-FILE.
27            05 EINGABE-GANZ            PIC X(130).
28
29        FD AUSGABE.
```

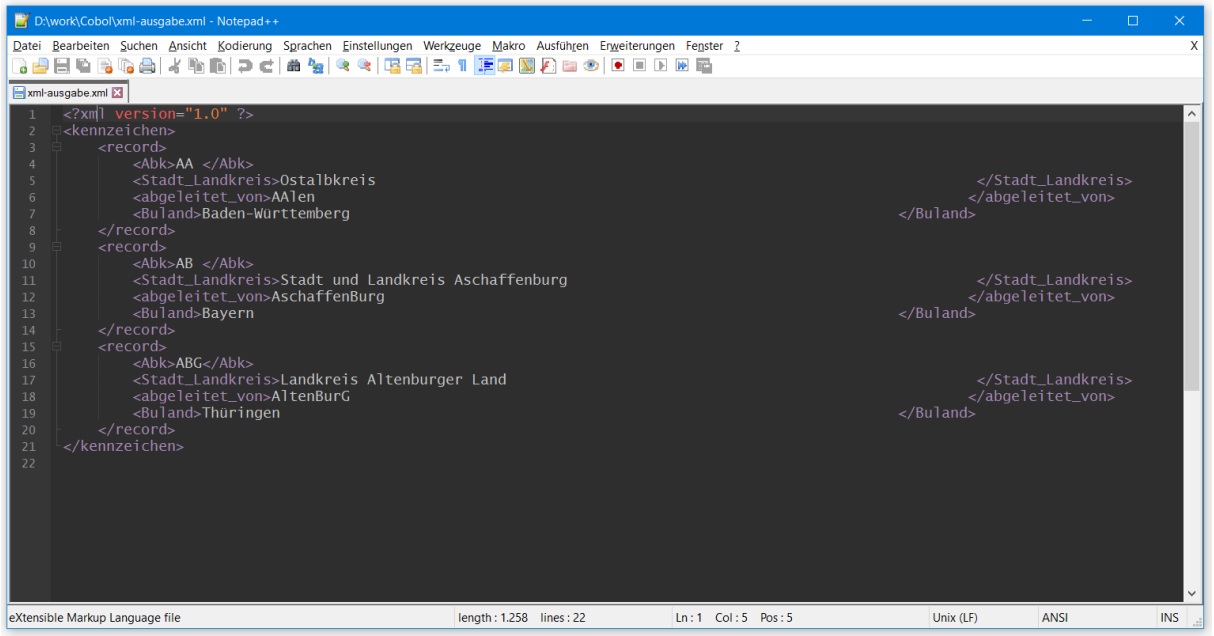
```
30         01 AUSGABE-FILE.
31             05 AUSGABE-ZEILE            PIC X(130).
32
33         WORKING-STORAGE SECTION.
34
35        *Definition der Schalter
36         01 EINGABE-ENDE-ERREICHT  PIC X(1).
37             88 EINGABE-ENDE-JA        VALUE "J".
38             88 EINGABE-ENDE-NEIN      VALUE "N".
39
40         01 FEHLER-SCHALTER        PIC X(1).
41             88 FEHLER-JA              VALUE "J".
42             88 FEHLER-NEIN            VALUE "N".
43
44         01 ERSTER-SATZ            PIC X(1).
45             88 ERSTER-SATZ-JA         VALUE "J".
46             88 ERSTER-SATZ-NEIN       VALUE "N".
47
48        *Definition der Variablen
49         01 VARIABLEN.
50             05 EINGABE-STATUS        PIC 9(2).
51             05 AUSGABE-STATUS        PIC 9(2).
52             05 WS-EINGABE            PIC X(130).
53             05 TEMP-AUSGABE-ZEILE    PIC X(130).
54             05 WS-INHALT-ABK         PIC X(3).
55             05 WS-INHALT-STADT       PIC X(80).
56             05 WS-INHALT-ABGEL       PIC X(80).
57             05 WS-INHALT-BULAND      PIC X(80).
58
59        *Definition der Konstanten
60         01 KONSTANTEN.
61             05 NUM-10                PIC 9(2) VALUE 10.
62             05 WS-AUSGABE.
63                10 AUS-VERSION.
64                   15 FILLER           PIC X(14) VALUE "<?xml version=".
65                   15 FILLER           PIC X(01) VALUE X'22'.
66                   15 FILLER           PIC X(03) VALUE "1.0".
67                   15 FILLER           PIC X(01) VALUE X'22'.
68                   15 FILLER           PIC X(02) VALUE "?>".
69                10 AUS-KENNZEICHEN-AUF PIC X(13) VALUE "<kennzeichen>".
70                10 AUS-KENNZEICHEN-ZU  PIC X(14) vALUE "</kennzeichen>".
71                10 AUS-RECORD_AUF      PIC X(13) VALUE "    <record>".
72                10 AUS-RECORD_ZU       PIC X(14) VALUE "    </record>".
73                10 AUS-ABK-AUF          PIC X(13) VALUE "        <Abk>".
74                10 AUS-ABK-ZU           PIC X(07) VALUE "</Abk>".
75                10 AUS-STADT-AUF        PIC X(25) VALUE
76                   "        <Stadt_Landkreis>".
77                10 AUS-STADT-ZU         PIC X(18) VALUE
78                   "</Stadt_Landkreis>".
79                10 AUS-ABGEL-AUF        PIC X(24) VALUE
80                   "        <abgeleitet_von>".
81                10 AUS-ABGEL-ZU         PIC X(17) VALUE
82                   "</abgeleitet_von>".
83                10 AUS-BULAND-AUF        PIC X(16) VALUE
84                   "        <Buland>".
85                10 AUS-BULAND-ZU         PIC X(17) VALUE
86                   "</Buland>".
```

```
 87            05 CHAR-SEMICOLON        PIC X(1) VALUE ";".
 88
 89         PROCEDURE DIVISION.
 90
 91        *Beginn der Steuerung
 92         STEUER SECTION.
 93            PERFORM INITIALISIERUNG
 94            PERFORM OEFFNEN-DATEIEN
 95            PERFORM LESEN-DATENSATZ
 96            PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA
 97                PERFORM KONVERTIERE-DATENSATZ
 98                PERFORM LESEN-DATENSATZ
 99            END-PERFORM
100            PERFORM ABSCHLUSSARBEITEN
101            PERFORM SCHLIESSEN-DATEIEN
102            STOP RUN.
103        *Ende der Steuerung
104
105        *Beginn der Prozeduren
106         INITIALISIERUNG SECTION.
107            DISPLAY "IN INITIALISIERUNG"
108
109        *    Setzen der Schalter auf Anfangszustand
110            SET EINGABE-ENDE-NEIN TO TRUE
111            SET FEHLER-NEIN TO TRUE
112            SET ERSTER-SATZ-JA TO TRUE
113
114        *    Alle Variablen auf einen Schlag initialisieren
115            INITIALIZE VARIABLEN
116         .
117         OEFFNEN-DATEIEN SECTION.
118            DISPLAY "IN OEFFNEN-DATEIEN"
119            OPEN INPUT EINGABE
120            OPEN OUTPUT AUSGABE
121         .
122         LESEN-DATENSATZ SECTION.
123            DISPLAY "IN LESEN-DATENSATZ"
124
125            READ EINGABE INTO WS-EINGABE
126               AT END SET EINGABE-ENDE-JA TO TRUE
127               NOT AT END DISPLAY EINGABE-GANZ
128            END-READ
129
130            EVALUATE EINGABE-STATUS
131               WHEN ZEROES
132               WHEN NUM-10
133                  CONTINUE
134               WHEN OTHER
135                  DISPLAY "EINGABE-STATUS = " EINGABE-STATUS
136                  SET FEHLER-JA TO TRUE
137            END-EVALUATE
138         .
139         KONVERTIERE-DATENSATZ SECTION.
140            DISPLAY "IN KONVERTIERE-DATENSATZ"
141
142            IF ERSTER-SATZ-JA
143                MOVE AUS-VERSION            TO TEMP-AUSGABE-ZEILE
```

```
144                 PERFORM SCHRIEBEN-DATENSATZ
145                 MOVE AUS-KENNZEICHEN-AUF    TO TEMP-AUSGABE-ZEILE
146                 PERFORM SCHRIEBEN-DATENSATZ
147                 SET ERSTER-SATZ-NEIN        TO TRUE
148             END-IF
149
150             MOVE AUS-RECORD_AUF             TO TEMP-AUSGABE-ZEILE
151             PERFORM SCHRIEBEN-DATENSATZ
152
153             UNSTRING WS-EINGABE DELIMITED BY CHAR-SEMICOLON
154                INTO WS-INHALT-ABK,
155                     WS-INHALT-STADT,
156                     WS-INHALT-ABGEL,
157                     WS-INHALT-BULAND
158             END-UNSTRING
159
160             STRING AUS-ABK-AUF
161                    WS-INHALT-ABK
162                    AUS-ABK-ZU
163             INTO TEMP-AUSGABE-ZEILE
164             PERFORM SCHRIEBEN-DATENSATZ
165
166             STRING AUS-STADT-AUF
167                    WS-INHALT-STADT
168                    AUS-STADT-ZU
169             INTO TEMP-AUSGABE-ZEILE
170             PERFORM SCHRIEBEN-DATENSATZ
171
172             STRING AUS-ABGEL-AUF
173                    WS-INHALT-ABGEL
174                    AUS-ABGEL-ZU
175             INTO TEMP-AUSGABE-ZEILE
176             PERFORM SCHRIEBEN-DATENSATZ
177
178             STRING AUS-BULAND-AUF
179                    WS-INHALT-BULAND
180                    AUS-BULAND-ZU
181             INTO TEMP-AUSGABE-ZEILE
182             PERFORM SCHRIEBEN-DATENSATZ
183
184             MOVE AUS-RECORD_ZU              TO TEMP-AUSGABE-ZEILE
185             PERFORM SCHRIEBEN-DATENSATZ
186             .
187
188         SCHRIEBEN-DATENSATZ SECTION.
189             DISPLAY "IN SCHRIEBEN-DATENSATZ"
190
191             WRITE AUSGABE-FILE FROM TEMP-AUSGABE-ZEILE
192
193             IF AUSGABE-STATUS > ZEROES
194                 DISPLAY "AUSGABE-STATUS = " AUSGABE-STATUS
195                 SET FEHLER-JA TO TRUE
196             END-IF
197
198             INITIALIZE TEMP-AUSGABE-ZEILE
199         .
200         ABSCHLUSSARBEITEN SECTION.
```

```
201             DISPLAY "IN ABSCHLUSSARBEITEN"
202             MOVE AUS-KENNZEICHEN-ZU         TO TEMP-AUSGABE-ZEILE
203             PERFORM SCHRIEBEN-DATENSATZ
204         .
205         SCHLIESSEN-DATEIEN SECTION.
206             DISPLAY "IN SCHLIESSEN-DATEIEN"
207             CLOSE EINGABE
208             CLOSE AUSGABE
209         .
210         END PROGRAM CSV-NACH-XML.
211
```

Und die Ausgabe aus dem Programm sollte dann so sein:

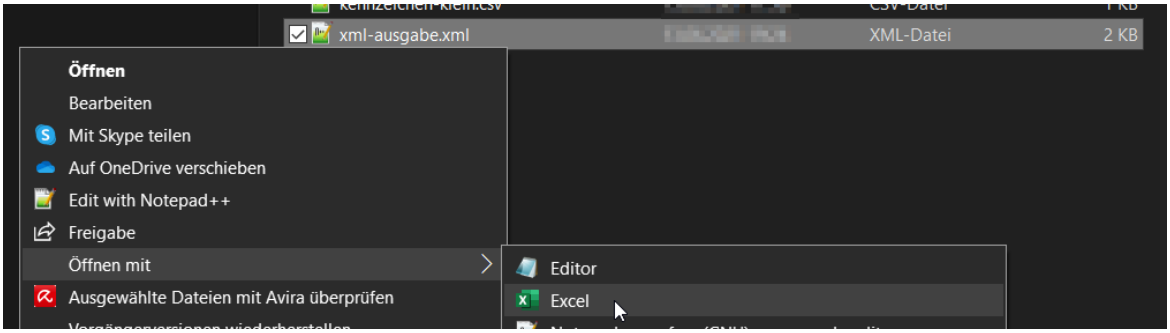


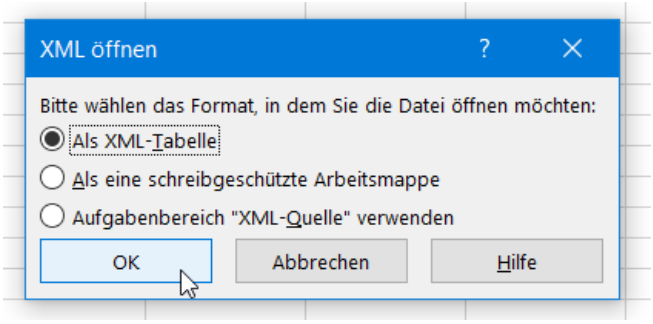
Sieht gut aus. An der Länge der Variablen Bestandteile werden wir später noch basteln.

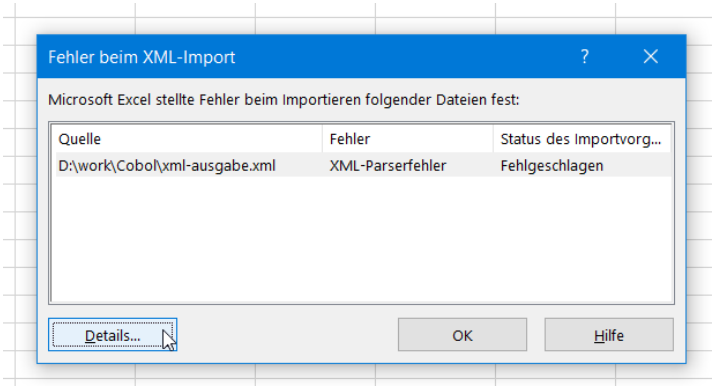
4.5. Umlaute behandeln

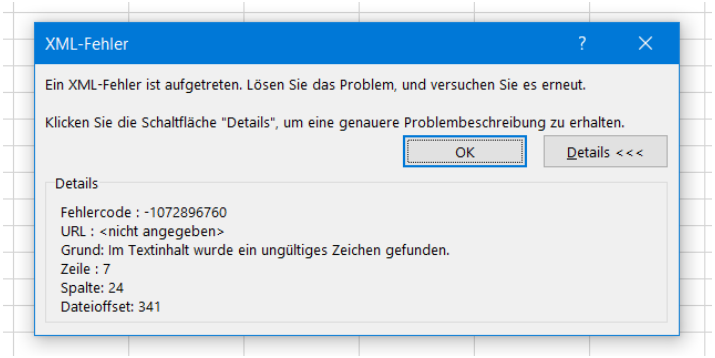
Sieht das nicht schon gut genug aus? Warum sollten wir die Umlaute noch behandeln müssen?

Machen wir die Probe. Eine xml-Datei kann von einem Tabellenkalkulationsprogramm eingelesen und als Tabelle ausgegeben werden. Bei mir gibt es bei dem Versuch die Datei in Excel zu öffnen, folgenden Hinweis:

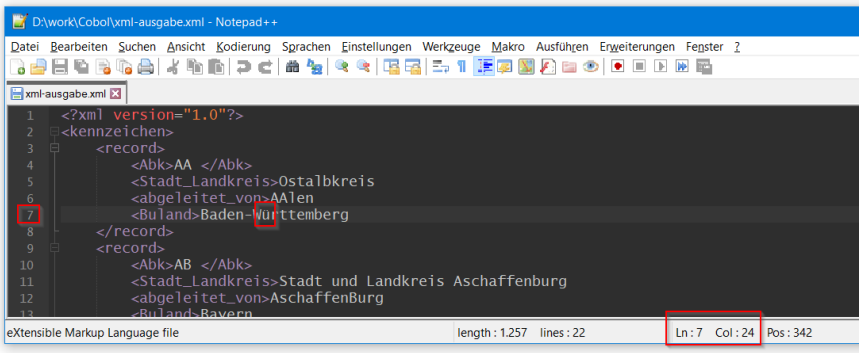






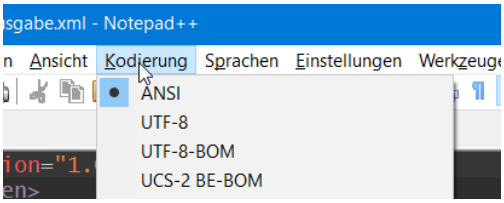


Das ungültige Zeichen soll in Zeile 7 Spalte 24 stehen. Schauen wir in die Datei rein, es ist das „ü“:

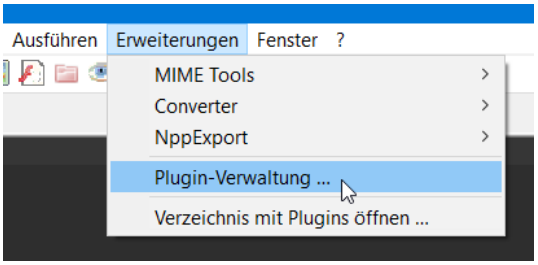


Damit können wir mit der von uns erstellten Datei (noch) nichts anfangen. Was aber ist das Problem? Schauen wir uns an, was im Editor zu sehen ist.

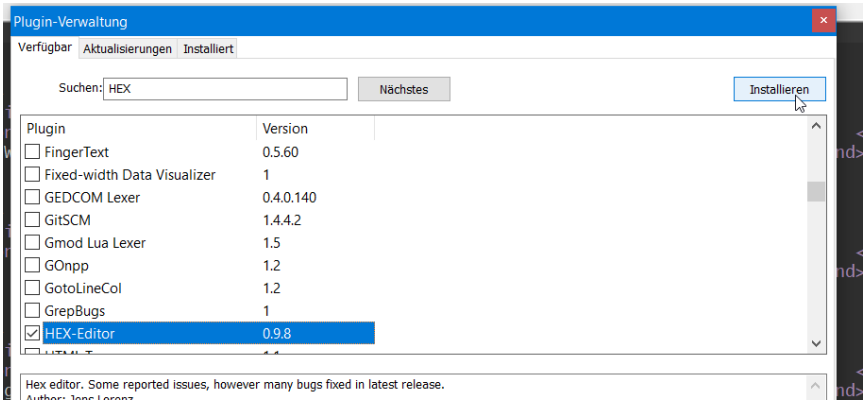
Wenn wir die xml-Datei öffnen, können wir uns die Kodierung, also den verwendeten Zeichensatz ansehen. In unserem Fall ist das ANSI:



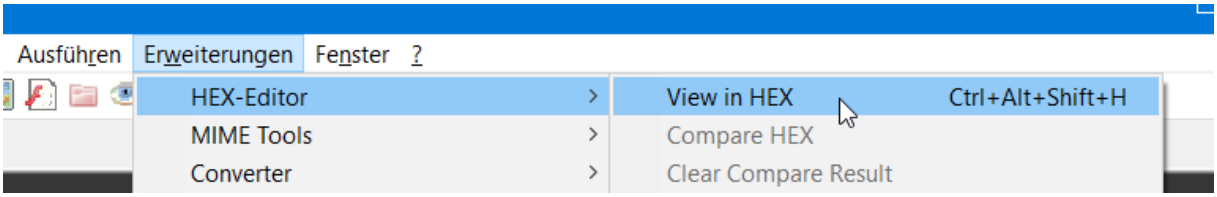
Für das Notepad++ gibt es eine Erweiterung, die wir hier sehr gut nutzen können, den HEX-Editor. Die Installation ist einfach, über „Erweiterungen/Plugin-Verwaltung…“



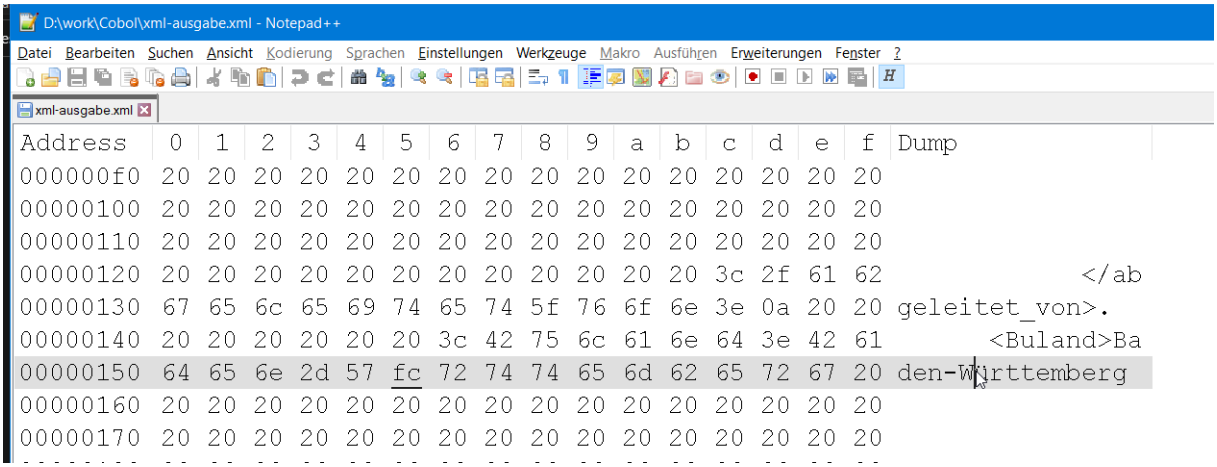
Öffnet sich ein Fenster, im Feld „Suchen:“ geben wir HEX ein und uns wird unter anderem der HEX-Editor angeboten. Markieren und mit „Installieren“ bestätigen:



Sobald die Erweiterung installiert ist, können wir uns die Inhalte der Datei über „Erweiterungen/HEX-Editor/View in HEX“ anschauen



Wenn wir uns vor dem „ü“ positionieren, sehen wir, dass der HEX-Wert „fc“ ist.

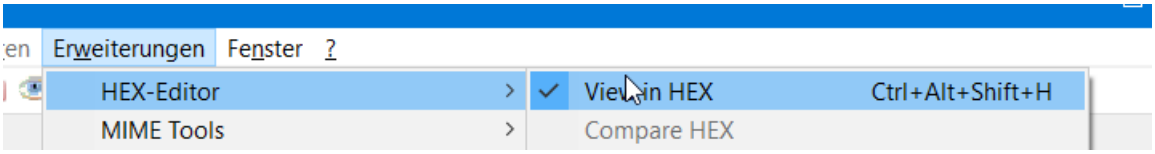


Das scheint also nicht richtig zu sein. Via Internetrecherche habe ich die HEX-Werte für das UTF-8 Format gesucht. Es ergibt sich eine Tabelle, wie wir welches Zeichen umzusetzen haben:

Zeichen	ANSI-HEX	UTF-8 HEX
Ä	c4	c3 84
Ö	d6	c3 96
Ü	dc	c3 9c
ä	e4	c3 a4
ö	f6	c3 b6
ü	fc	c3 bc
ß	df	c3 9f

Okay, unser „ü“ in der xml-Datei ist also mit „fc“ codiert, wir müssen ein Byte einschieben (das „c3“) und aus dem „fc“ ein „bc“ machen.

Machen wir das zuerst in der xml-Datei. Im HEX-Editor können wir keine Zeichen einfügen, also müssen wir da wieder raus:



Fügen wir jetzt dem ü in Baden-Württemberg einfach ein weiteres ü hinzu - Baden-Wüürttemberg und gehen wieder in den HEX-Editor:

xml-ausgabe.xml

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
000000f0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000100	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000110	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000120	20	20	20	20	20	20	20	20	20	20	20	20	3c	2f	61	62	</ab
00000130	67	65	6c	65	69	74	65	74	5f	76	6f	6e	3e	0a	20	20	geleitet_von>.
00000140	20	20	20	20	20	20	3c	42	75	6c	61	6e	64	3e	42	61	<Buland>Ba
00000150	64	65	6e	2d	57	fc	fc	72	74	74	65	6d	62	65	72	67	den-WüürttemberG
00000160	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

Wenn wir jetzt das erste „ü“ in „c3“ ändern und das zweite „ü“ in „bc“ sollten wir das Notwendige erreicht haben:

xml-ausgabe.xml

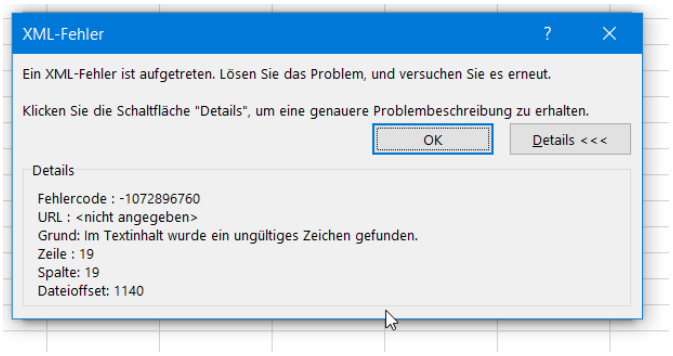
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
000000f0	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000100	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000110	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000120	20	20	20	20	20	20	20	20	20	20	20	20	3c	2f	61	62	</ab
00000130	67	65	6c	65	69	74	65	74	5f	76	6f	6e	3e	0a	20	20	geleitet_von>.
00000140	20	20	20	20	20	20	3c	42	75	6c	61	6e	64	3e	42	61	<Buland>Ba
00000150	64	65	6e	2d	57	c3	bc	72	74	74	65	6d	62	65	72	67	den-WÃ¼rttemberG
00000160	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	

Sieht komisch aus, oder? Machen wir den HEX-Editor wieder zu. Sieht immer noch nicht besser aus:

xml-ausgabe.xml

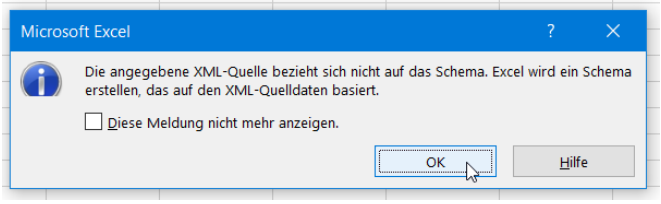
```
1  <?xml version="1.0"?>
2  <kennzeichen>
3      <record>
4          <Abk>AA </Abk>
5          <Stadt_Landkreis>Ostalbkreis
6          <abgeleitet_von>AAlen
7          <Buland>Baden-WÃ¼rttemberg
8      </record>
```

Machen wir die Probe aufs Exempel, die xml-Datei speichern und dann wie oben versuchen mit einem Tabellenkalkulationsprogramm zu öffnen:



Aha, über das „ü“ in Baden-Württemberg ist die Import-Funktion hinweggekommen, jetzt ist Zeile 19 Spalte 19 ein Problem – das „ü“ in Thüringen.

Also ändern wir auch das genauso wie das erste „ü“ und speichern die Datei. Neuer Versuch:

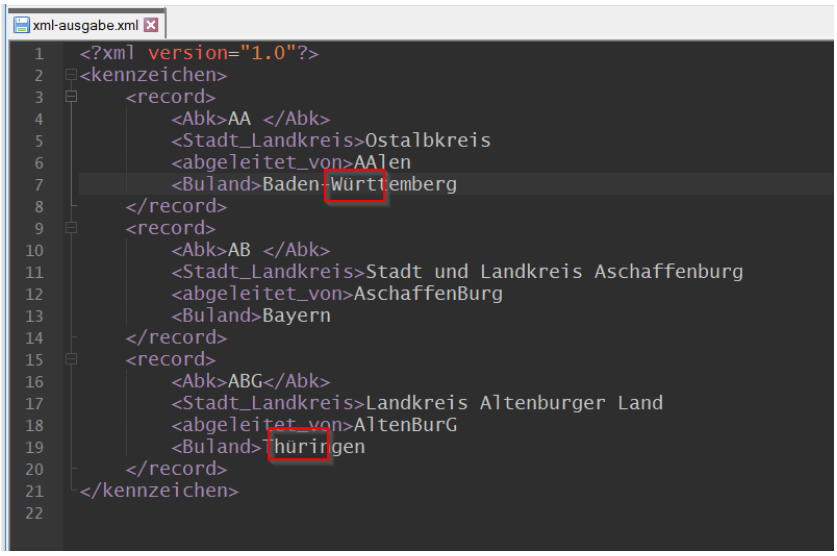


Mit OK bestätigen und siehe da:

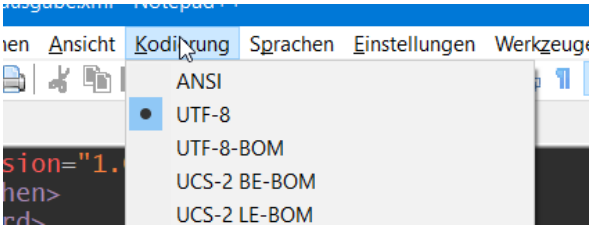
	A	B	C	D
1	Abk	Stadt_Landkreis	abgeleitet_von	Buland
2	AA	Ostalbkreis	AAlen	Baden-Württemberg
3	AB	Stadt und Landkreis Aschaffenburg	AschaffenBurg	Bayern
4	ABG	Landkreis Altenburger Land	AltenBurG	Thüringen
5				

Yes!! Also haben wir das richtig gemacht!

Und zur Bestätigung – wenn wir die xml-Datei im Notepad++ erneut öffnen, sieht auf einmal alles gut aus:



Wie kommt das? Schauen wir auf die Kodierung, die hat jetzt gewechselt:



Notepad++ hat also die Codierung beim Öffnen der Datei erkannt und automatisch gesetzt. Damit haben wir aber die Bestätigung, dass dieses Vorgehen okay ist.

Was bedeutet das jetzt für unser Programm? Zunächst einmal die Definition von jeder Menge Konstanten. Bei mir sieht das so aus:

```
*    EINGEHENDE ZEICHEN
*    GROSSES Ä
     05 HEXA-C4                    PIC X(01) VALUE X'C4'.
*    GROSSES Ü
     05 HEXA-DC                    PIC X(01) VALUE X'DC'.
*    GROSSES Ö
     05 HEXA-D6                    PIC X(01) VALUE X'D6'.
*    KLEINES ä
     05 HEXA-E4                    PIC X(01) VALUE X'E4'.
*    KLEINES ü
     05 HEXA-FC                    PIC X(01) VALUE X'FC'.
*    KLEINES ö
     05 HEXA-F6                    PIC X(01) VALUE X'F6'.
*    DAS SCHARFE S
     05 HEXA-DF                    PIC X(01) VALUE X'DF'.
*    FILLER, ALLEN ZEICHEN IN DER AUSGABE VORANGESTELLT
     05 HEXA-C3                    PIC X(01) VALUE X'c3'.
*    AUSGEHENDE ZEICHEN
*    GROSSES Ä
     05 HEXA-84                    PIC X(01) VALUE X'84'.
*    GROSSES Ü
     05 HEXA-9C                    PIC X(01) VALUE X'9c'.
*    GROSSES Ö
     05 HEXA-96                    PIC X(01) VALUE X'96'.
*    KLEINES ä
     05 HEXA-A4                    PIC X(01) VALUE X'a4'.
*    KLEINES ü
     05 HEXA-BC                    PIC X(01) VALUE X'bc'.
*    KLEINES ö
     05 HEXA-B6                    PIC X(01) VALUE X'b6'.
*    DAS SCHARFE S
     05 HEXA-9F                    PIC X(01) VALUE X'9F'.
```

Wir müssen jetzt Zeichen für Zeichen der Eingabe durchgehen und das eingehende Zeichen gegen das ausgehende Zeichen austauschen.

Um das zu erreichen bedienen wir uns einer internen Tabelle. Schauen wir uns zuerst wieder den generellen Aufbau von internen Tabellen an:

```
01  TABELLE-1 OCCURS 20 TIMES INDEXED BY TAB-1-IND.
    05 VORNAME                   PIC X(20).
    05 NACHNAME                  PIC X(40).
    05 SCHUHGROESSE              PIC 9(2).
```

Die Interne Tabelle heißt TABELLE-1 und hat 20 Zeilen. Das ist wieder festgelegt, wir erinnern uns, mit „variable“ hat es COBOL nicht so. Die Tabelle besteht aus 3 Spalten, Vorname, Nachname und Schuhgröße. Soll ein bestimmter Satz angesprochen werden, müssen wir den Index `TAB-1-IND` zuerst auf einen bestimmten Wert setzen und dann können wir das Feld mit den Index in Klammern ansprechen:

```
SET TAB-1-IND to 17
DISPLAY "Vorname aus Zeile 17 ist: " VORNAME(TAB-1-IND)
```

In unserem Fall bauen wir uns eine Tabelle mit nur einem Byte Inhalt und das lassen wir 80 mal vorkommen:

```
01  TABELLE-1 OCCURS 80 TIMES INDEXED BY TAB-1-IND.
    05 EIN-BYTE                  PIC X(1).
```

Leider können wir die Tabelle nicht einfach mit der MOVE-Anweisung befüllen, wir brauchen hier den Kunstgriff des „REDEFINES“. Mit der REDEFIENS-Anweisung binden wir 2 Felder aneinander. Wenn in dem einen Feld etwas passiert, wird es automatisch im anderen nachgezogen. Bei uns sieht das dann so aus:

```
*interne Tabellen inklusive des REDEFINES-Feldes
 01  RED-EIN-FELD                PIC X(80).
 01  TABELLE-1 REDEFINES RED-EIN-FELD
              OCCURS 80 TIMES INDEXED BY TAB-1-IND.
     05 EIN-BYTE                  PIC X(1).
```

Jetzt können wir die MOVE-Anweisung ausführen:

```
     MOVE WS-INHALT-STADT         TO RED-EIN-FELD
```

Die Anweisung REDEFINES sorgt wie beschrieben automatisch dafür, dass wir auf eine gefüllte Tabelle zugreifen können.

Und jetzt kommt der Trick. Wir definieren uns eine zweite Tabelle, die wir Feld für Feld aus der ersten Tabelle füllen, aber wenn es ein Umlaut gibt, ersetzen wir an der originalen Position den HEX-Wert gegen „c3“ und gehen dann in der Ausgabe-Tabelle ein Feld weiter und setzen dort das zweite Byte wie gewünscht. Auch die zweite Tabelle hat ein REDEFINES, nur wirkt es in die andere Richtung. Damit können wir die Inhalte der Tabelle wieder in einem Stück in die Ausgabe schreiben.

```
…
    05 WS-TEMP-AUSGABE           PIC X(80).
…

01  RED-AUS-FELD                PIC X(80).
01  TABELLE-2 REDEFINES RED-AUS-FELD
              OCCURS 80 TIMES INDEXED BY TAB-2-IND.
    05 AUS-BYTE                  PIC X(1).
```

Das klingt kompliziert, wenn wir es durchgehen seht Ihr was ich meine. Ich mache das erst einmal für das „ü“ im Feld Bundesland (`WS-INHALT-BULAND`).

Ausgangspunkt ist die UNSTRING-Anweisung. Nachdem UNSTRING durch ist, steht im Feld der WS-INHALT-BULAND „Baden-Württemberg“ mit einem „ü“ das hexadezimal „bc“ ist.

Im Anschluss an die UNSTRING-Anweisung übergeben wir den Inhalt an das Feld `RED-EIN-FELD`, die REDEFINES-Anweisung sorgt dafür, dass der Inhalt der `TABELLE-1` ebenfalls „Baden-Württemberg“ ist.

Damit das jetzt schön übersichtlich ist, bauen wir den Aufruf einer neuen SECTION ein– der `UMLAUTE-UMSETZEN SECTION:`

```
            …
            END-UNSTRING

            MOVE WS-INHALT-BULAND          TO RED-EIN-FELD
            PERFORM UMLAUTE-UMSETZEN

            STRING AUS-ABK-AUF
            …
```

Die SECTION selbst sieht dann so aus, die Erklärungen habe ich als Kommentare eingebaut:

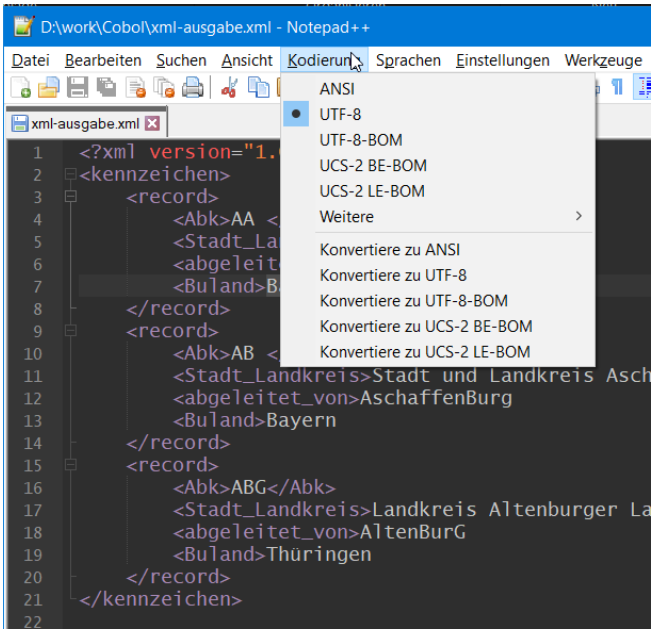
```
        UMLAUTE-UMSETZEN SECTION.
            DISPLAY "IN UMLAUTE-UMSETZEN"

       *    Initialisieren des Index für die 2. Tabelle:
            SET TAB-2-IND TO 0

       *    Schleife um die 1. Tabelle:
            PERFORM VARYING TAB-1-IND FROM 1 BY 1
                    UNTIL TAB-1-IND > 80

       *        Index der 2. Tabelle erhöhen:
                SET TAB-2-IND UP BY 1
       *        Übergabe Inhalte von einer Tabelle zur anderen:
                MOVE EIN-BYTE(TAB-1-IND) TO AUS-BYTE(TAB-2-IND)

       *        Behandlung kleines ü
                IF AUS-BYTE(TAB-2-IND) = HEXA-FC
       *            Ersetzen HEX fc durch c3
                    MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
       *            Index um 1 erhöhen um in das nächste Feld zu gelangen
                    SET TAB-2-IND UP BY 1
       *            Ersetzen leer durch HEX bc
                    MOVE HEXA-BC TO AUS-BYTE(TAB-2-IND)
                END-IF

       *    Ende der Schleife um die 1. Tabelle:
            END-PERFORM

       *    Das Feld RED-AUS-FELD beinhaltet die Werte Tabelle-2,
       *    daher kann das jetzt an das Feld WS-TEMP-AUSGABE
       *    übergeben werden:
            DISPLAY "RED-AUS-FELD = " RED-AUS-FELD
            MOVE RED-AUS-FELD TO WS-TEMP-AUSGABE
        .
```

Damit haben wir den neuen Wert im Feld `WS-TEMP-AUSGABE` gespeichert. Der Aufruf geht jetzt zurück in die KONVERTIEREN SECTION, dort müssen wir noch die STRING-Anweisung für das BULAND ändern.

```
            STRING AUS-BULAND-AUF
                   WS-TEMP-AUSGABE
                   AUS-BULAND-ZU
            INTO TEMP-AUSGABE-ZEILE
```

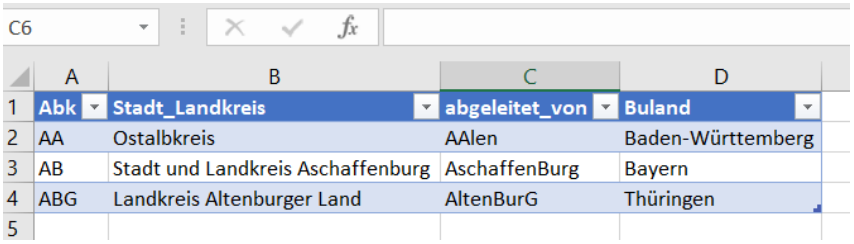
Dann sind wir hier erstmal fertig und können das testen. Speichern, Cygwin starten, in Verzeichnis wechseln, cobc -x aufrufen, mit ./ laufen lassen.

Die nach dem Lauf erzeugte xml-Datei können wir jetzt im Notepad++ öffnen und uns die Kodierung anschauen:



UTF-8, keine Überraschung, alles richtig gemacht.

Wenn wir die xml-Datei wieder im Tabellenkalkulationsprogramm öffnen, sehen wir auch hier alles in Ordnung:



Was auffällt ist, dass die ganzen Leerzeichen alle herausgefiltert werden. Es ist also nicht notwendig, die 80 Stellen auf die richtige Anzahl zu reduzieren. Eigentlich… Im letzten Kapitel spreche ich das nochmal an.

Damit ist bewiesen, dass die Umwandlung in dieser Form erfolgen kann. Daher weiten wir das jetzt auf alle 4 Felder aus.

Jetzt ist Fleißarbeit gefragt, die anderen Konstanten müssen eingebaut, transformiert, zurück-gegeben und geschrieben werden.

Zur Erleichterung hier der fertige Code für dieses Kapitel:

```
1       ******************************************************************
2       * Author:    papa
3       * Date:      Juni 2021
4       * Purpose:   Konvertieren von Inhalten aus csv nach xml
5       *            Ausbaustufe 4.5. Umlaute behandeln
6       * Tectonics: cobc
```

```
 7       ****************************************************************
 8        IDENTIFICATION DIVISION.
 9        PROGRAM-ID. CSV-NACH-XML.
10        ENVIRONMENT DIVISION.
11        INPUT-OUTPUT SECTION.
12        FILE-CONTROL.
13
14        SELECT EINGABE ASSIGN TO 'kennzeichen-klein.csv'
15        ORGANIZATION IS LINE SEQUENTIAL
16        FILE STATUS IS EINGABE-STATUS.
17
18        SELECT AUSGABE ASSIGN TO 'xml-ausgabe.xml'
19        ORGANIZATION IS LINE SEQUENTIAL
20        FILE STATUS IS AUSGABE-STATUS.
21
22        DATA DIVISION.
23        FILE SECTION.
24
25        FD EINGABE.
26        01 EINGABE-FILE.
27            05 EINGABE-GANZ             PIC X(130).
28
29        FD AUSGABE.
30        01 AUSGABE-FILE.
31            05 AUSGABE-ZEILE            PIC X(130).
32
33        WORKING-STORAGE SECTION.
34
35       *Definition der Schalter
36        01 EINGABE-ENDE-ERREICHT  PIC X(1).
37            88 EINGABE-ENDE-JA        VALUE "J".
38            88 EINGABE-ENDE-NEIN      VALUE "N".
39
40        01 FEHLER-SCHALTER        PIC X(1).
41            88 FEHLER-JA              VALUE "J".
42            88 FEHLER-NEIN            VALUE "N".
43
44        01 ERSTER-SATZ            PIC X(1).
45            88 ERSTER-SATZ-JA         VALUE "J".
46            88 ERSTER-SATZ-NEIN       VALUE "N".
47
48       *Definition der Variablen
49        01 VARIABLEN.
50            05 EINGABE-STATUS        PIC 9(2).
51            05 AUSGABE-STATUS        PIC 9(2).
52            05 WS-EINGABE            PIC X(130).
53            05 TEMP-AUSGABE-ZEILE    PIC X(130).
54            05 WS-INHALT-ABK         PIC X(3).
55            05 WS-INHALT-STADT       PIC X(80).
56            05 WS-INHALT-ABGEL       PIC X(80).
57            05 WS-INHALT-BULAND      PIC X(80).
58            05 WS-TEMP-AUSGABE       PIC X(80).
59
60       *Definition der Konstanten
61        01 KONSTANTEN.
62            05 NUM-10                PIC 9(2) VALUE 10.
63            05 WS-AUSGABE.
```

```
64              10 AUS-VERSION.
65                 15 FILLER           PIC X(14) VALUE "<?xml version=".
66                 15 FILLER           PIC X(01) VALUE X'22'.
67                 15 FILLER           PIC X(03) VALUE "1.0".
68                 15 FILLER           PIC X(01) VALUE X'22'.
69                 15 FILLER           PIC X(02) VALUE "?>".
70              10 AUS-KENNZEICHEN-AUF PIC X(13) VALUE "<kennzeichen>".
71              10 AUS-KENNZEICHEN-ZU  PIC X(14) vALUE "</kennzeichen>".
72              10 AUS-RECORD_AUF      PIC X(13) VALUE "    <record>".
73              10 AUS-RECORD_ZU       PIC X(14) VALUE "    </record>".
74              10 AUS-ABK-AUF          PIC X(13) VALUE "        <Abk>".
75              10 AUS-ABK-ZU           PIC X(07) VALUE "</Abk>".
76              10 AUS-STADT-AUF        PIC X(25) VALUE
77                 "        <Stadt_Landkreis>".
78              10 AUS-STADT-ZU         PIC X(18) VALUE
79                 "</Stadt_Landkreis>".
80              10 AUS-ABGEL-AUF        PIC X(24) VALUE
81                 "        <abgeleitet_von>".
82              10 AUS-ABGEL-ZU         PIC X(17) VALUE
83                 "</abgeleitet_von>".
84              10 AUS-BULAND-AUF        PIC X(16) VALUE
85                 "        <Buland>".
86              10 AUS-BULAND-ZU         PIC X(17) VALUE
87                 "</Buland>".
88           05 CHAR-SEMICOLON        PIC X(1) VALUE ";".
89       *   EINGEHENDE ZEICHEN
90       *   GROSSES Ä
91           05 HEXA-C4                   PIC X(01) VALUE X'C4'.
92       *   GROSSES Ü
93           05 HEXA-DC                   PIC X(01) VALUE X'DC'.
94       *   GROSSES Ö
95           05 HEXA-D6                   PIC X(01) VALUE X'D6'.
96       *   KLEINES ä
97           05 HEXA-E4                   PIC X(01) VALUE X'E4'.
98       *   KLEINES ü
99           05 HEXA-FC                   PIC X(01) VALUE X'FC'.
100      *   KLEINES ö
101          05 HEXA-F6                   PIC X(01) VALUE X'F6'.
102      *   DAS SCHARFE S
103          05 HEXA-DF                   PIC X(01) VALUE X'DF'.
104      *   FILLER, ALLEN ZEICHEN IN DER AUSGABE VORANGESTELLT
105          05 HEXA-C3                   PIC X(01) VALUE X'c3'.
106      *   AUSGEHENDE ZEICHEN
107      *   GROSSES Ä
108          05 HEXA-84                   PIC X(01) VALUE X'84'.
109      *   GROSSES Ü
110          05 HEXA-9C                   PIC X(01) VALUE X'9c'.
111      *   GROSSES Ö
112          05 HEXA-96                   PIC X(01) VALUE X'96'.
113      *   KLEINES ä
114          05 HEXA-A4                   PIC X(01) VALUE X'a4'.
115      *   KLEINES ü
116          05 HEXA-BC                   PIC X(01) VALUE X'bc'.
117      *   KLEINES ö
118          05 HEXA-B6                   PIC X(01) VALUE X'b6'.
119      *   DAS SCHARFE S
120          05 HEXA-9F                   PIC X(01) VALUE X'9F'.
```

```
121
122       *interne Tabellen inklusive des REDEFINES-Feldes
123        01  RED-EIN-FELD                 PIC X(80).
124        01  TABELLE-1 REDEFINES RED-EIN-FELD
125                   OCCURS 80 TIMES INDEXED BY TAB-1-IND.
126            05 EIN-BYTE                  PIC X(1).
127        01  RED-AUS-FELD                 PIC X(80).
128        01  TABELLE-2 REDEFINES RED-AUS-FELD
129                   OCCURS 100 TIMES INDEXED BY TAB-2-IND.
130            05 AUS-BYTE                  PIC X(1).
131
132        PROCEDURE DIVISION.
133
134       *Beginn der Steuerung
135        STEUER SECTION.
136            PERFORM INITIALISIERUNG
137            PERFORM OEFFNEN-DATEIEN
138            PERFORM LESEN-DATENSATZ
139            PERFORM WITH TEST BEFORE UNTIL EINGABE-ENDE-JA OR FEHLER-JA
140                PERFORM KONVERTIERE-DATENSATZ
141                PERFORM LESEN-DATENSATZ
142            END-PERFORM
143            PERFORM ABSCHLUSSARBEITEN
144            PERFORM SCHLIESSEN-DATEIEN
145            STOP RUN.
146       *Ende der Steuerung
147
148       *Beginn der Prozeduren
149        INITIALISIERUNG SECTION.
150            DISPLAY "IN INITIALISIERUNG"
151
152       *    Setzen der Schalter auf Anfangszustand
153            SET EINGABE-ENDE-NEIN TO TRUE
154            SET FEHLER-NEIN TO TRUE
155            SET ERSTER-SATZ-JA TO TRUE
156
157       *    Alle Variablen auf einen Schlag initialisieren
158            INITIALIZE VARIABLEN
159        .
160        OEFFNEN-DATEIEN SECTION.
161            DISPLAY "IN OEFFNEN-DATEIEN"
162            OPEN INPUT EINGABE
163            OPEN OUTPUT AUSGABE
164        .
165        LESEN-DATENSATZ SECTION.
166            DISPLAY "IN LESEN-DATENSATZ"
167
168            READ EINGABE INTO WS-EINGABE
169               AT END SET EINGABE-ENDE-JA TO TRUE
170               NOT AT END DISPLAY EINGABE-GANZ
171            END-READ
172
173            EVALUATE EINGABE-STATUS
174               WHEN ZEROES
175               WHEN NUM-10
176                  CONTINUE
177               WHEN OTHER
```

```
178                 DISPLAY "EINGABE-STATUS = " EINGABE-STATUS
179                 SET FEHLER-JA TO TRUE
180             END-EVALUATE
181         .
182         KONVERTIERE-DATENSATZ SECTION.
183             DISPLAY "IN KONVERTIERE-DATENSATZ"
184
185             IF ERSTER-SATZ-JA
186                 MOVE AUS-VERSION            TO TEMP-AUSGABE-ZEILE
187                 PERFORM SCHRIEBEN-DATENSATZ
188                 MOVE AUS-KENNZEICHEN-AUF    TO TEMP-AUSGABE-ZEILE
189                 PERFORM SCHRIEBEN-DATENSATZ
190                 SET ERSTER-SATZ-NEIN        TO TRUE
191             END-IF
192
193             MOVE AUS-RECORD_AUF             TO TEMP-AUSGABE-ZEILE
194             PERFORM SCHRIEBEN-DATENSATZ
195
196             UNSTRING WS-EINGABE DELIMITED BY CHAR-SEMICOLON
197                INTO WS-INHALT-ABK,
198                     WS-INHALT-STADT,
199                     WS-INHALT-ABGEL,
200                     WS-INHALT-BULAND
201             END-UNSTRING
202
203             MOVE WS-INHALT-ABK        TO RED-EIN-FELD
204             PERFORM UMLAUTE-UMSETZEN
205
206             STRING AUS-ABK-AUF
207                    WS-TEMP-AUSGABE
208                    AUS-ABK-ZU
209             INTO TEMP-AUSGABE-ZEILE
210             PERFORM SCHRIEBEN-DATENSATZ
211
212             MOVE WS-INHALT-STADT      TO RED-EIN-FELD
213             PERFORM UMLAUTE-UMSETZEN
214
215             STRING AUS-STADT-AUF
216                    WS-TEMP-AUSGABE
217                    AUS-STADT-ZU
218             INTO TEMP-AUSGABE-ZEILE
219             PERFORM SCHRIEBEN-DATENSATZ
220
221             MOVE WS-INHALT-ABGEL      TO RED-EIN-FELD
222             PERFORM UMLAUTE-UMSETZEN
223
224             STRING AUS-ABGEL-AUF
225                    WS-TEMP-AUSGABE
226                    AUS-ABGEL-ZU
227             INTO TEMP-AUSGABE-ZEILE
228             PERFORM SCHRIEBEN-DATENSATZ
229
230             MOVE WS-INHALT-BULAND     TO RED-EIN-FELD
231             PERFORM UMLAUTE-UMSETZEN
232
233             STRING AUS-BULAND-AUF
234                    WS-TEMP-AUSGABE
```

```
235                 AUS-BULAND-ZU
236            INTO TEMP-AUSGABE-ZEILE
237            PERFORM SCHRIEBEN-DATENSATZ
238
239            MOVE AUS-RECORD_ZU           TO TEMP-AUSGABE-ZEILE
240            PERFORM SCHRIEBEN-DATENSATZ
241            .
242
243         UMLAUTE-UMSETZEN SECTION.
244            DISPLAY "IN UMLAUTE-UMSETZEN"
245
246        *    Initialisieren des Index für die 2. Tabelle:
247            SET TAB-2-IND TO 0
248
249        *    Schleife um die 1. Tabelle:
250            PERFORM VARYING TAB-1-IND FROM 1 BY 1
251                    UNTIL TAB-1-IND > 80
252
253        *        Index der 2. Tabelle erhöhen:
254                 SET TAB-2-IND UP BY 1
255        *        Übergabe Inhalte von einer Tabelle zur anderen:
256                 MOVE EIN-BYTE(TAB-1-IND) TO AUS-BYTE(TAB-2-IND)
257
258        *        Behandlung großes Ä
259                 IF AUS-BYTE(TAB-2-IND) = HEXA-C4
260                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
261                     SET TAB-2-IND UP BY 1
262                     MOVE HEXA-84 TO AUS-BYTE(TAB-2-IND)
263                 END-IF
264
265        *        Behandlung großes Ö
266                 IF AUS-BYTE(TAB-2-IND) = HEXA-D6
267                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
268                     SET TAB-2-IND UP BY 1
269                     MOVE HEXA-96 TO AUS-BYTE(TAB-2-IND)
270                 END-IF
271
272        *        Behandlung großes Ü
273                 IF AUS-BYTE(TAB-2-IND) = HEXA-DC
274                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
275                     SET TAB-2-IND UP BY 1
276                     MOVE HEXA-9C TO AUS-BYTE(TAB-2-IND)
277                 END-IF
278
279        *        Behandlung kleines ä
280                 IF AUS-BYTE(TAB-2-IND) = HEXA-E4
281                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
282                     SET TAB-2-IND UP BY 1
283                     MOVE HEXA-A4 TO AUS-BYTE(TAB-2-IND)
284                 END-IF
285
286        *        Behandlung kleines ö
287                 IF AUS-BYTE(TAB-2-IND) = HEXA-F6
288                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
289                     SET TAB-2-IND UP BY 1
290                     MOVE HEXA-B6 TO AUS-BYTE(TAB-2-IND)
291                 END-IF
```

```
292
293        *        Behandlung kleines ü
294                 IF AUS-BYTE(TAB-2-IND) = HEXA-FC
295        *            Ersetzen HEX fc durch c3
296                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
297        *            Index um 1 erhöhen um in das nächste Feld zu gelangen
298                     SET TAB-2-IND UP BY 1
299        *            Ersetzen leer durch HEX bc
300                     MOVE HEXA-BC TO AUS-BYTE(TAB-2-IND)
301                 END-IF
302
303        *        Behandlung scharfes s (ß)
304                 IF AUS-BYTE(TAB-2-IND) = HEXA-DF
305                     MOVE HEXA-C3 TO AUS-BYTE(TAB-2-IND)
306                     SET TAB-2-IND UP BY 1
307                     MOVE HEXA-9F TO AUS-BYTE(TAB-2-IND)
308                 END-IF
309
310        *    Ende der Schleife um die 1. Tabelle:
311             END-PERFORM
312
313        *    Das Feld RED-AUS-FELD beinhaltet die Werte Tabelle-2,
314        *    daher kann das jetzt an das Feld WS-TEMP-AUSGABE
315        *    übergeben werden:
316             DISPLAY "RED-AUS-FELD = " RED-AUS-FELD
317             MOVE RED-AUS-FELD TO WS-TEMP-AUSGABE
318            .
319
320            SCHRIEBEN-DATENSATZ SECTION.
321                 DISPLAY "IN SCHRIEBEN-DATENSATZ"
322
323                 WRITE AUSGABE-FILE FROM TEMP-AUSGABE-ZEILE
324
325                 IF AUSGABE-STATUS > ZEROES
326                     DISPLAY "AUSGABE-STATUS = " AUSGABE-STATUS
327                     SET FEHLER-JA TO TRUE
328                 END-IF
329
330                 INITIALIZE TEMP-AUSGABE-ZEILE
331            .
332            ABSCHLUSSARBEITEN SECTION.
333                 DISPLAY "IN ABSCHLUSSARBEITEN"
334                 MOVE AUS-KENNZEICHEN-ZU         TO TEMP-AUSGABE-ZEILE
335                 PERFORM SCHRIEBEN-DATENSATZ
336            .
337            SCHLIESSEN-DATEIEN SECTION.
338                 DISPLAY "IN SCHLIESSEN-DATEIEN"
339                 CLOSE EINGABE
340                 CLOSE AUSGABE
341            .
342            END PROGRAM CSV-NACH-XML.
343
```

Da ist bis jetzt ja schon ganz schön viel Code zusammengekommen, sehr schön. Probiert das auch mit der großen Datei aus. Dazu Zeile 14 ändern. Statt

```
14         SELECT EINGABE ASSIGN TO 'kennzeichen_klein.csv'
```

die andere Datei angeben, bei mir kennzeichen.csv:

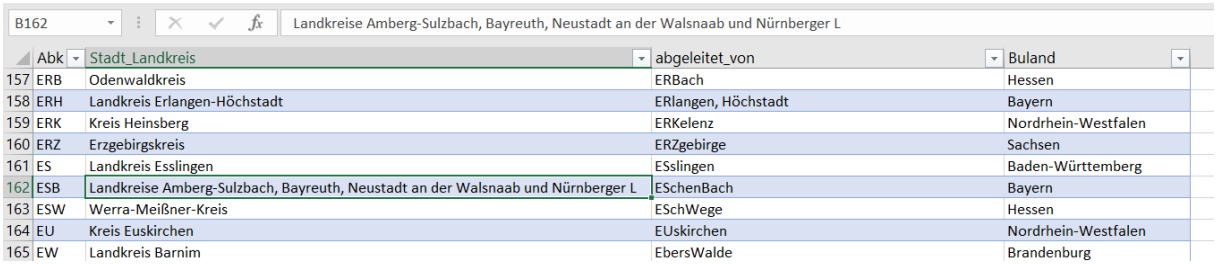
```
14         SELECT EINGABE ASSIGN TO 'kennzeichen.csv'
```

Das Ergebnis ist dann genauso überprüfbar wie die kleine Datei.

In Kapitel 4.2.2 haben wir die Länge der Eingabedatei anhand der beiden längsten Einträge gemacht. Das ist immer noch valide, wir haben keine Probleme mit abgeschnittenen Inhalten.

Anders sieht es mit unseren getroffenen Annahmen in Kapitel 4.4 aus.

In Zeile 162 in der großen Datei steht im Feld „Stadt_Landkreis“ der Text „Landkreise Amberg-Sulzbach, Bayreuth, Neustadt an der Walsnaab und Nürnberger Land“. Das sind 82 Stellen. Wie man sieht, werden die letzten 3 Stellen abgeschnitten. Unschön…



B162 | Landkreise Amberg-Sulzbach, Bayreuth, Neustadt an der Walsnaab und Nürnberger L

	Abk	Stadt_Landkreis	abgeleitet_von	Buland
157	ERB	Odenwaldkreis	ERBach	Hessen
158	ERH	Landkreis Erlangen-Höchstadt	ERlangen, Höchstadt	Bayern
159	ERK	Kreis Heinsberg	ERKelenz	Nordrhein-Westfalen
160	ERZ	Erzgebirgskreis	ERZgebirge	Sachsen
161	ES	Landkreis Esslingen	ESslingen	Baden-Württemberg
162	ESB	Landkreise Amberg-Sulzbach, Bayreuth, Neustadt an der Walsnaab und Nürnberger L	ESchenBach	Bayern
163	ESW	Werra-Meißner-Kreis	ESchWege	Hessen
164	EU	Kreis Euskirchen	EUskirchen	Nordrhein-Westfalen
165	EW	Landkreis Barnim	EbersWalde	Brandenburg

Aber warum 3 Stellen und nicht 2? 80 bis 82 sind doch nur 2 Stellen? Stimmt, aber da ist ja noch ein „ü“ in „Nürnberger“. Da müssen wir aus der zweiten internen Tabelle noch einen draufrechnen. Geht schnell sich da zu verhauen, gelle?

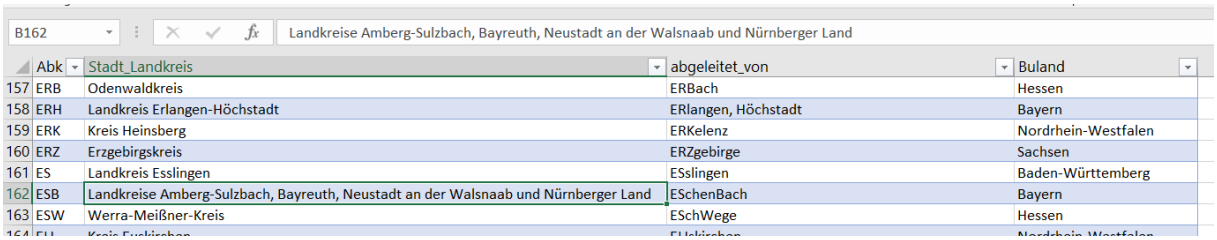
Je nachdem wie viel Arbeit Ihr bereit seid in das Projekt noch zu stecken, gibt es eine schnelle und mehrere schöne Lösungen.

Die „schnelle Lösung“ sieht so aus, dass wir alle Deklaration auf irgendwas größer oder gleich 83 anpassen. Betroffen sind:

```
 55            05 WS-INHALT-STADT       PIC X(82).
 56            05 WS-INHALT-ABGEL        PIC X(82).
 57            05 WS-INHALT-BULAND       PIC X(82).
 58            05 WS-TEMP-AUSGABE        PIC X(82).
    …
122        *interne Tabellen inklusive des REDEFINES-Feldes
123         01  RED-EIN-FELD                 PIC X(82).
124         01  TABELLE-1 REDEFINES RED-EIN-FELD
125                       OCCURS 82 TIMES INDEXED BY TAB-1-IND.
126            05 EIN-BYTE                   PIC X(1).
127         01  RED-AUS-FELD                 PIC X(83).
128         01  TABELLE-2 REDEFINES RED-AUS-FELD
129                       OCCURS 83 TIMES INDEXED BY TAB-2-IND.
130            05 AUS-BYTE                   PIC X(1).
    …
```

```
243         UMLAUTE-UMSETZEN SECTION.
    …
250             PERFORM VARYING TAB-1-IND FROM 1 BY 1
251                     UNTIL TAB-1-IND > 83
    …
```

Das Ergebnis ist dann okay:



B162 | Landkreise Amberg-Sulzbach, Bayreuth, Neustadt an der Walsnaab und Nürnberger Land

	Abk	Stadt_Landkreis	abgeleitet_von	Buland
157	ERB	Odenwaldkreis	ERBach	Hessen
158	ERH	Landkreis Erlangen-Höchstadt	ERlangen, Höchstadt	Bayern
159	ERK	Kreis Heinsberg	ERKelenz	Nordrhein-Westfalen
160	ERZ	Erzgebirgskreis	ERZgebirge	Sachsen
161	ES	Landkreis Esslingen	ESslingen	Baden-Württemberg
162	ESB	Landkreise Amberg-Sulzbach, Bayreuth, Neustadt an der Walsnaab und Nürnberger Land	ESchenBach	Bayern
163	ESW	Werra-Meißner-Kreis	ESchWege	Hessen

Die schönere Variante wäre, für jedes der 4 Felder eine eigene maximale Länge zu ermitteln. Dann bräuchten wir auch je Ausgabefeld eine eigene interne Tabelle die dann so groß ist, wie das längste Feld. Das ist allerdings ein größerer Aufwand und es stellt sich die Frage, ob der angesichts des Nutzens betrieben werden soll, denn aktuell ist die xml-Datei ja schon nutzbar.

Die aus meiner Sicht allerschönste Variante wäre aber, wenn es, wie in unserem Fall, nur ein Ausgabefeld gibt, und das immer die Länge hat, die gerade gebraucht wird. Aktuell ist die Länge immer 83, obwohl die Abkürzung maximal 3stellig ist.

Damit sind wir hier am Ende angelangt, die Anwendung macht was sie soll. Ich hoffe, es hat Euch Spaß gemacht und Ihr konntet einen ersten Überblick über COBOL gewinnen.

4.6. Abschluss Projekt

Wie in allen bisher zusammen gestellten Dokumentationen hier der Ausblick, wie es weiter gehen könnte.

Dem alten COBOL-Entwickler in mir gehen einige Dinge gegen den Strich und die Ehre, iach werde noch eine optimierte Version hinterherschieben, das ist dann aber ein eigenes kleine Projekt und mehr eine Herzensangelegenheit als eine Notwendigkeit. Dafür gibt es mehrere Gründe.

Es wird ein Haufen an Programmlaufzeit verbraten für unnötige Aufrufe. So wird die Schleife um die erste Tabelle (Zeilen 250/251) immer 82 mal ausgeführt. Für das Feld „ABK“ müsste es nur maximal 3 mal durchgeführt werden, 79 mal läuft das also umsonst. Und das bei über 700 Einträgen.

Außerdem sind die Ergebnisse der DISPLAY-Anweisungen nach dem Programmlauf nicht mehr zugänglich. Besser wäre es, statt der Ausgabe der Meldung auf der Konsole via DISPLAY, eine zusätzliche Ausgabedatei zu definieren, die dann als Log-Datei fungieren würde.

Mittlerweile ist auch die Namensgebung der einzelnen SECTIONs nicht mehr sehr übersichtlich, da gibt es auch Möglichkeiten, das besser zu strukturieren.

Also viel Verbesserungspotenzial.

Das war es jetzt aber von meiner Seite, ich hoffe, es hat Euch gefallen. Bei Fragen und Anregungen, aber auch, wenn Euch Fehler auffallen, würde ich mich über Rückmeldung über papa@papa-programmiert.de freuen.

Viel Spaß beim Programmieren!