RPG and Unicode – Teil 1
Definition: EBCDIC, CCSID, ASCII und Unicode

Bereits seit Release 7.2 können ins RPG UTF-8-Daten verarbeitet werden. Bei Texten in klassischem ASCII oder EBCDIC-Code entspricht ein Zeichen genau einem Byte. Im Gegensatz dazu kann bei UTF-8 die Länge eines Zeichens zwischen einem und vier Byte schwanken und das kann dazu führen, dass Informationen verloren gehen können oder auch, dass die Texte bzw. die zugrundeliegenden hexadezimalen Werte falsch interpretiert werden. Bevor wir uns damit beschäftigen, wie in RPG-Probleme beim Einsatz von UTF-8 vermieden werden können, zunächst einmal eine Begriffsdefinition.

Schauen wir uns zunächst einmal die Daten an und wie sie gespeichert werden.

Code Tabellen

Daten werden im Binär-Code (also Nullen und Einsen) in Bits hinterlegt. Jedes einzelne Bit kann 2 Zustände einnehmen, also 0 oder 1. Acht Bits werden wiederum zu einem Byte zusammengefasst, wobei ursprünglich ein Byte einem Zeichen entsprochen hat. Damit können in einem Byte bis zu 256 (2^8) unterschiedliche Zeichen gespeichert werden.

Nur welcher Hex-Code entspricht welchem Zeichen?

Schauen wir uns einfach einmal x‘7D‘ an:

- ' — Wird der Hex-Wert innerhalb eines Textes im EBCDIC-Code verwendet, so entspricht x‘7D‘ dem Hochkomma (').
- } — Wird der Hex-Wert innerhalb eines Textes im ASCII Code oder in UTF-8 verwendet, so entspricht x‘7D‘ der schließenden geschweiften Klammer (}).
- -7 — Befindet sich der Hex-Wert in einer gepackt numerischen Zahl, so entspricht x‘7D‘ der negativen Zahl -7.
- 125 — Handelt es sich bei dem Byte um einen ganzzahligen Wert (1-Byte Integer), so entspricht x‘7D‘ der Zahl 125.

Wie aus der vorherigen Auflistung klar wird, benötigt es Regeln und Code-Tabellen, die es erlauben einen Hex-Wert „richtig“ zu interpretieren.

Wenn wir auf der IBM i (und auch auf anderen IBM-Maschinen) arbeiten, werden die alphanumerischen Daten per Default im EBCDIC-Code mit einer entsprechenden sprachenspezifischen CCSID verwaltet und gespeichert.

Nun ist die IBM i keine Insel mehr und wir müssen in der Lage sein Daten von anderen Systemen und aus dem Web zu übernehmen und richtig zu interpretieren … und an dieser Stelle wird KEIN EBCDIC, sondern ASCII oder Unicode (UTF-8) Code verwendet.

Deshalb zunächst einmal einen Überblick über die gängigen Codes:

ASCII Code - American Code for Information Interchange

Der ASCII-Code wurde der 1963 für den Austausch von Daten über Fernschreiber entwickelt.

Ein Zeichen entspricht einem Byte. Allerdings werden nicht alle 8 Bits eines Bytes für die Darstellung von Zeichen verwendet, sondern nur 7, während das 8. Bit als sogenanntes Prüf-Bit verwendet wurde. Damit können beim ASCII-Code 128 (2^7) unterschiedliche Zeichen pro Byte verwaltet werden. Das war so lange ausreichend, so lange nur die Buchstaben A-Z in Groß- und Kleinschreibung, Ziffern von 0-9 und ein paar wenige Sonder- und Steuerzeichen zum Austausch von Daten verwendet wurden.

EBCDIC Code - Extended Binary Coded Decimal Interchange Code

Der EBCDIC-Code (Extended Binary Coded Decimal Interchange Code) wurde annähernd zeitgleich (mit dem ASCII-Code) von der IBM für (ihre) Großrechner-Systeme entwickelt. Auch heute noch werden auf der IBM i alphanumerische Zeichenketten per Default im EBCDIC Code verwaltet.

RPG and Unicode – Teil 1
Definition: EBCDIC, CCSID, ASCII und Unicode

Im Gegensatz zum ASCII-Code werden beim EBCDIC-Code alle 8 Bits zur Darstellung eines Zeichens verwendet. Damit konnte der EBCDIC doppelt so viele unterschiedliche Zeichen ($2^8 = 256$) wie der ASCII-Code für ein Byte verwalten.

ASCII wurde hauptsächlich im PC-Bereich eingesetzt während auf den IBM Maschinen der EBCDIC-Code verwendet wird.

Da der ASCII und der EBCDIC-Code unabhängig voneinander entwickelt wurden, werden für die gleichen Zeichen nicht unbedingt die gleichen Hex-Werte verwendet. Somit sind für die Konvertierung von EBCDIC nach ASCII und umgekehrt Umsetzungstabellen erforderlich.

Mit zunehmender Globalisierung und dem Austausch von Daten, die in unterschiedlichen Sprachen und Schriftsystemen dargestellt werden, sind weder 128 noch 256 verschiedene Zeichen ausreichend.

Um Texte in unterschiedlichen Sprachen korrekt verarbeiten zu können, hat man sowohl bei ASCII als auch bei EBCDIC Code Pages eingeführt, die jeweils die Besonderheiten einer Sprache oder Sprachfamilie (z.B. akzentuierte Buchstaben) beinhalten.

Dem EBCDIC-Code werden unterschiedlichen Code-Tabellen über die CCSID (Character Set Id) zugeordnet, z.B. 37 = amerikanisches Englisch, 273 = deutsch für Deutschland und Österreich.

Das funktioniert so lange gut, solange man mit einer einzigen Sprache bzw. mit einem einzigen Zeichenset arbeiten kann. Wenn jedoch in einer Spalte z.B. deutsche, polnische, griechische und kyrillische Adressen verwaltet werden sollen, wird es schon komplizierter. … und wenn dann auch Texte in unterschiedlichen Sprachen in der gleichen Spalte gespeichert werden sollen, wird es schwierig.

UCS-2 - Universal Coded Character Set

Eine Möglichkeit dem Problem abzuhelfen ist, anstatt eines Single-Byte-Character-Set (1 Byte=1 Zeichen) ein Double-Byte-Character-Set (2 Byte = 1 Zeichen) zu verwenden.

Ende der 1980ger Jahre wurde UCS-2 entwickelt.

UCS-2 ist eine Fixed-Byte-Kodierung: jedes Zeichen ist genau 2 Byte lang. UCS-2 kann damit die 65.536 Zeichen der Basic Multilingual Plane (BMP) darstellen. Damit können fast alle Zeichen in den Schriften der lebenden Sprachen, sowie die gängigen Sonderzeichen im gleichen Character Set verwaltet werden.

UTF-8 - 8-Bit UCS Transformation Format

UTF-8 wurde 1992 von Ken Thompson und Rob Pike UTF-8 (8-Bit UCS Transformation Format) eingeführt, das sich zu der am weitesten verbreitete Kodierung für Unicode-Zeichen entwickelt hat.

Außerdem hat sich UTF-8 als De-facto-Standard-Zeichenkodierung des Internets und damit verbundener Dokumenttypen etabliert. Annähernd alle HTML, JSON und XML-Daten werden in UTF-8 codiert und ausgetauscht.

Bei der UTF-8-Kodierung wird jedem Unicode-Zeichen eine Byte-Kette mit einer Länge von zwischen einem und vier Byte zugeordnet. Damit lassen sich – wie bei allen UTF-Formaten – alle Unicode-Zeichen abbilden.

In der Code-Tabelle von UTF-8 entsprechen die Hex-Werte der ersten 128 Zeichen den ASCII-Zeichen. Das betrifft alle Ziffern, sowie die Groß- und Klein-Buchstaben des lateinischen Grund-Alphabets. Diese Zeichen werden weiterhin in einem Byte gesichert.

Zusätzliche Zeichen in europäischen Sprachen mit lateinischer Schrift, z. B. ä, ß, é, ł, werden durch zwei Byte dargestellt. Griechische, kyrillische oder arabische Zeichen belegen ebenfalls zwei Bytes. Zeichen aus indischen und fernöstlichen Schriften belegen meist drei Byte, einige seltene Zeichen und Schriften sogar vier Byte je Zeichen.

Damit kann jedes beliebige Zeichen im UTF-8 Format dargestellt werden.

UTF-16

Bei der UTF-16-Kodierung wird jedem Unicode-Codepunkt eine speziell kodierte Kette von ein oder zwei 16-Bit-Einheiten zugeordnet, d. h. von zwei oder vier Bytes, so dass sich – wie auch bei den anderen UTF-Formaten – alle Unicode-Zeichen abbilden lassen.

Während UTF-8 eine zentrale Bedeutung in Internet-Protokollen hat, wird UTF-16 vielerorts zur internen Repräsentation von Zeichenketten verwendet, z. B. in aktuellen Versionen von .Net-Framework, Java und Tcl.

Soweit zu einer kurzen Übersicht über die unterschiedlichen alphanumerischen Codes. Auf der IBM i wird weiterhin vorranging im EBCDIC-Code gearbeitet. Für den Datenaustausch mit anderen Systemen müssen die Daten immer öfter in UTF-8 konvertiert werden, bzw. UTF-8-Daten müssen korrekt verarbeitet werden können.

EBCDIC Code und CCSID (Character Set Id)

Solange wir auf der IBM i im Single-Byte-Character-Set und im EBCDIC-Code arbeiten, gibt es keine Probleme so lange nur mit einer einzigen Sprache gearbeitet wird. Des Weiteren gibt es auch keine Probleme, wenn mit unterschiedlichen Sprachen gearbeitet wird und nur die Zeichen A-Z (Groß- und Kleinbuchstaben) und Ziffern verwendet werden. Sobald jedoch die in den einzelnen Sprachen unterschiedlichen akzentuierten Buchstaben dazukommen, wird es mit 256 unterschiedlichen Zeichen eng, ganz zu schweigen von kyrillischen, griechischen, arabischen, japanischen Schriftzeichen.

Aus diesem Grund hat man zusätzlich zum EBCDIC Code Character Set Ids (CCSID) eingeführt. Bei den CCSIDs für Sprachen, die mit lateinischem Alphabet geschrieben werden, haben die Zeichen, die überall verwendet werden (A-Z, a-z, 0-9 sowie gängige Sonderzeichen) jeweils den gleichen Hex-Code. Die für die unterschiedlichen Sprachen benötigten akzentuierten Zeichen, werden auf den „freien“ Hex-Werten eingeordnet. Dabei kann das gleiche Zeichen in unterschiedlichen CCSIDs unterschiedliche Hex-Werte haben und auch der gleiche Hex-Wert kann in unterschiedlichen CCSIDs mit anderen Zeichen belegt sein.

In dem folgenden Beispiel wird ein Text mit diversen akzentuierten Buchstaben in unterschiedlichen CCSID ausgegeben und jeweils der entsprechende Hex-Wert angezeigt.

```
Values(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1141), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1141))), -- Deutsch
      (Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1142), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1142))), -- Dänemark, Norwegen
      (Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1143), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1143))), -- Finnland Schweden
      (Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1144), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1144))), -- Italien
      (Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1145), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1145))), -- Spanien
      (Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1147), Hex(Cast('ABC_äöüß_éàëø' as VarChar(20) CCSID 1147))); -- Frankreich
```

00001	00002
ABC_äöüß_éàëø	C1C2C36DC06AD0A16D51445370
ABC_äöüß_éàëø	C1C2C36D43CCA1596D5144536A
ABC_äöüß_éàëø	C1C2C36DC06AA1596D79445370
ABC_äöüß_éàëø	C1C2C36D43CCDC596D5AC05370
ABC_äöüß_éàëø	C1C2C36D43CCDC596D51445370
ABC_äöüß_éàëø	C1C2C36D43CCDC596DC07C5370

Abbildung 1 - CCSID - Hex-Werte

Mit zunehmender Internationalisierung muss man sich diesen Problemen stellen. Wobei, einfach CCSID 65535 (also KEINE CCSID) zu verwenden und die Daten einfach so wie sie kommen sichern, ist die schlechteste Lösung.

Eine Lösung wäre alles auf Double-Byte-Character-Set umzustellen, das kann auch RPG korrekt verarbeiten.

Eine andere Lösung wäre, insbesondere, da die Daten mehr und mehr im UTF-8-Format erstellt, übertragen und verarbeitet werden müssen, direkt auf UTF-8 umzustellen … nur dabei tut sich RPG schwer!

RPG and Unicode – Teil 1
Definition: EBCDIC, CCSID, ASCII und Unicode

In den folgenden Artikel werden wir uns mit der Konvertierung und der Verarbeitung von UTF-8-Daten in RPG beschäftigen.

Bis dahin schon einmal viel Spass mit den unterschiedlichen alphanumerischen Codes.