Codepages

Definitions, Some Implications

Business Programming 1 | **Business Programming 2**

REXX → **ooRexx** → **BSF4ooRexx**

Basics, Parsing	Commands, APIs	Window-Automatisation, Web-Scripting	Security, Debugging	Graphical User Interfaces (GUI), Sockets, ...

Prof. Rony G. Flatscher

7-Bit Codepage: ASCII

- Codepage: encoding of characters in a specific manner
- **ASCII** codepage (cf. https://en.wikipedia.org/wiki/ASCII)
  - "American Standard Code for Information Interchange "
  - **7-Bit** code allows for *128* different codepoints (*2**7*)
    - Numbered from "0" through "127" (decimal) or "00" through "7F" (hexadecimal)
    - First 32 characters (decimal "0" through "31", hexadecimal "00" through "1F") and last character (decimal "127", hexadecimal "7F") defined for "control characters", also dubbed "non-printable characters"
      - e.g. for telex machines and typewriters like "bell" (decimal "7", hexadecimal "07"), "carriage return" (decimal "13", hexadecimal "0D"), "line feed" (decimal "10", hexadecimal "0A"), "horizontal tabulator" (decimal "09", hexadecimal "09"), ...
    - All other codepoints are for "printable characters" including space
  - American English standard, hence no non-English characters!

8-Bit Codepages: DOS and Windows

- **8-bit** (*2**8*) doubles available codepoints from 128 to 256 !
  - A computer *byte* consists of 8 *bits*, hence able to represent an 8-bit character
  - The additional 128 codepoints can be used for assigning codepoints to e.g. German characters, box/drawing characters, Greek characters and more ...
    - However, worldwide there are many, many more characters than codepoints available!
  - Computer companies like IBM or Microsoft defined various 8-bit codepages
    - The first 128 codepoints *may* be defined to be the ASCII encoding
    - The second 128 codepoints got used for characters for spefic regions, e.g.
      - DOS Codepage **437**: ASCII plus Western European and special characters
        - Cf. https://en.wikipedia.org/wiki/Code_page_437
      - DOS Codepage **850**: ASCII plus Western European characters and special characters
        - Cf. https://en.wikipedia.org/wiki/Code_page_850
      - Windows Codepage **1252**: ASCII plus Western European characters and special characters
        - Cf. https://en.wikipedia.org/wiki/Windows-1252

8-Bit Codepages: Encoding Problems, 1

- Some characters may not be available at all in a certain codepage
- The same characters may be placed at different codepoints in different codepages
  - Example: encoding the lowercase German umlaut "**ü**"
    - DOS Codepage **437**/**850**: codepoint "129" (decimal), hexadecimal "81"
    - Windows Codepage **1252**: codepoint "252" (decimal), hexadecimal "FC"
    - Text with German umlauts encoded in one codepage may not display the expected German umlaut characters in a different codepage!
  - In general all characters in the upper 128 codepoints of an 8-bit codepage can only be displayed (processed) correctly if using the same codepage

8-Bit Codepages: Encoding Problems, 2

```
say "umlaut-u in 437/850 codepage has codepoint 129 (81 hex):"
do cp over 437, 850, 1252
   address system "chcp" cp
   say "Codepage" cp":" "hex 81:" 81~x2c "decimal: 129:" 129~d2c
end
say

say "umlaut-u in 1252 codepage has codepoint 252 (FC hex):"
do cp over 437, 850, 1252
   address system "chcp" cp
   say "Codepage" cp":" "hex FC:" FC~x2c "decimal: 252:" 252~d2c
end
```

Output (Windows):

```
umlaut-u in 437/850 codepage has codepoint 129 (81 hex):
Active code page: 437
Codepage 437: hex 81: ü decimal: 129: ü
Active code page: 850
Codepage 850: hex 81: ü decimal: 129: ü
Active code page: 1252
Codepage 1252: hex 81:  decimal: 129:

umlaut-u in 1252 codepage has codepoint 252 (FC hex):
Active code page: 437
Codepage 437: hex FC: ⁿ decimal: 252: ⁿ
Active code page: 850
Codepage 850: hex FC: ³ decimal: 252: ³
Active code page: 1252
Codepage 1252: hex FC: ü decimal: 252: ü
```

Microsoft Word and Cp1252

- In the Western world MS Word seems to encode in codepage **1252**
- All text to be read from and written to MS Word: codepage **1252** !
- If input text got encoded for a different codepage, one *must* convert the text from that codepage to **1252**
- Use the public routine `bsf.iconv(text,fromCodepage,toCodepage)` from BSF.CLS to *reliably* convert from one codepage to another
  - E.g. if text was encoded in codepage 850 then invoke it as

    ```
    text1252=bsf.iconv(text, "cp850", "cp1252")
    ```
  - If converting from/to UTF-8 (unicode) use `"utf-8"` as codepage argument

Example: Microsoft Word and bsf.iconv()

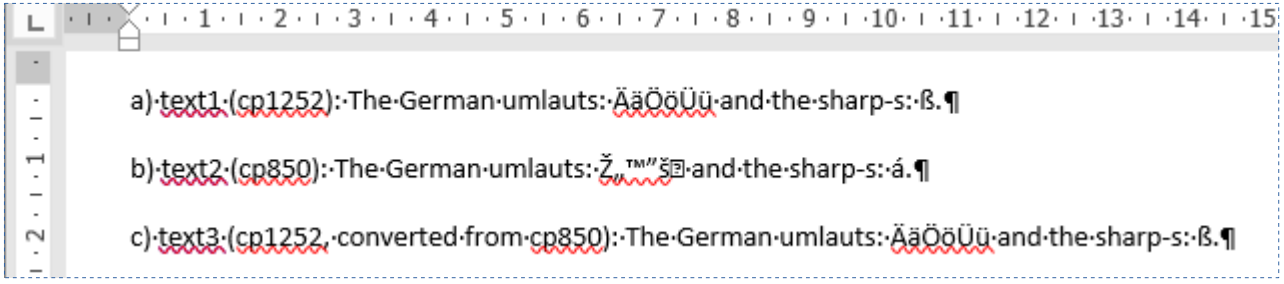
```
word = .OLEObject~New("word.Application")
word~Visible = .true                    -- make word visible
document = word~documents~add           -- add new document
textEncoding=document~textEncoding
say "word's text encoding:" textEncoding
Selection = word~selection
text1 = "The German umlauts: ÄäÖöÜü and the sharp-s: ß."  -- cp1252
selection~~typeText("a) text1 (cp1252):" text1)~~typeParagraph

text2 = "The German umlauts: Ž„™”š and the sharp-s: á."  -- cp850
selection~~typeText("b) text2 (cp850):"  text2)~~typeParagraph

text3=bsf.iconv(text2,"cp850","cp1252")   -- convert text from 850 to 1252
selection~~typeText("c) text3 (cp1252, converted from cp850):"  text3)~~typeParagraph
word~quit

::requires "BSF.CLS"    -- get ooRexx-Java bridge
```

```
word's text encoding: 1252
```



Prof. Rony G. Flatscher

BSF.Clipboard Class from BSF.CLS

- Makes it easy to copy images and strings to the system clipboard
- Makes it easy to paste images and strings from the system clipboard
- Option to explicitly state the codepage to use for strings
  - setString(string *[, encodedInCodepage]*)
    - Allows to indicate the codepage that was used for the string
    - The clipboard will get the string in Unicode
  - getString(*[encodeWithCodepage]*)
    - Allows to indicate the codepage that should be used to encode the returned string
- Other useful methods of the BSF.Clipboard class:
  - isEmpty (returns .true or .false), clear (empties the system clipboard), getDataFlavours (returns an array of Java DataFlavor objects indicating the types), setImage(image) and getImage (returns a java.awt.Image)

Prof. Rony G. Flatscher

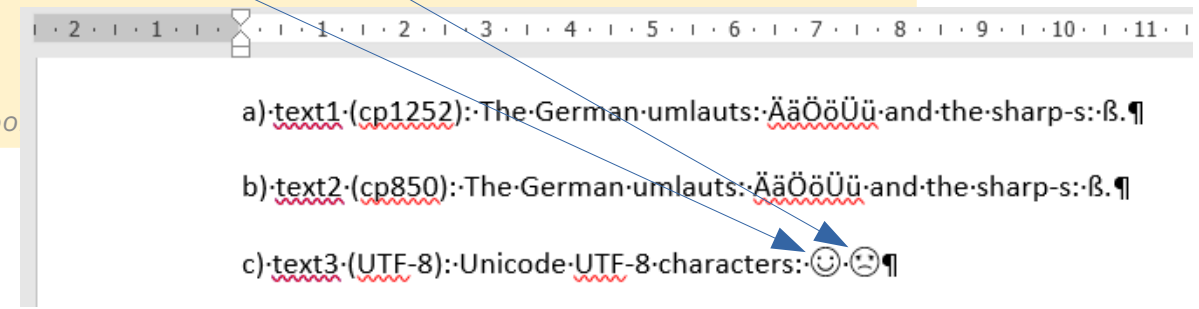
Example: Microsoft Word and Bsf.Clipboard

```
word = .OLEObject~New("word.Application")
word~Visible = .true                    -- make word visible
document = word~documents~add           -- add new document
textEncoding=document~textEncoding
say "word's text encoding:" textEncoding
Selection = word~selection
text1 = "The German umlauts: ÄäÖöÜü and the sharp-s: ß."  -- cp1252
.bsf.clipBoard~setString("a) text1 (cp1252):" text1, "CP1252")
selection~~paste~~typeParagraph

text2 = "The German umlauts: Ž„™”š and the sharp-s: á."  -- cp850
.bsf.clipBoard~setString("b) text2 (cp850):" text2, "cp850")
selection~~paste~~typeParagraph
      -- UTF-8 encoded emoticons: | smiley:   | frownie:
text3= "Unicode UTF-8 characters:" "e2 98 ba"x "e2 98 b9"x  -- UTF-8
.bsf.clipBoard~setString("c) text3 (UTF-8):" text3, "UTF-8")
selection~~paste~~typeParagraph
word~quit

::requires "BSF.CLS"    -- get oo
```

```
word's text encoding: 1252
```



Further Information ...

- Starting with ooRexx 5.1 see the WindowsClipboard class
  - Documentation book "Windows Extensions Reference" (winextensions.pdf)
    - Methods copy and paste and their Unicode related samples
- Overview and description of the many existing 8-bit codepages
  - Cf. https://en.wikipedia.org/wiki/Code_page
- Unicode
  - Multibyte encodings (between one and four bytes per character!)
    - UTF-7, UTF-8, UTF-16, UTF-16LE, UTF-16BE, UTF-32, UTF-32LE, UTF-32BE
  - Can represent all characters of any living and dead language in the world!
  - Homepage of the Unicode organisation developing the standard
    - https://home.unicode.org/
    - Another overview: https://en.wikipedia.org/wiki/Unicode