

デブキャンアンコールセッション

「Delphiでの文字コードのハンドリングについて」

有限会社 エイブル
富永 英明



EMBARCADERO
TECHNOLOGIES.



はじめに

まずは文字コードの種類を知る

- 日本で使われる有名所の文字コード
 - Shift_JIS
 - ISO-2022-JP
 - EUC-JP
 - UTF-8 (Unicode)
 - UTF-16 (Unicode)
- まずは文字コードがどうなっているのか知らない事には始まらない。
- 今回のセッション内容は...
 - 第16回デブキャン【1E】+ “課外授業” + 第11回デブキャン【A6】の一部 + α

文字集合

Unicode

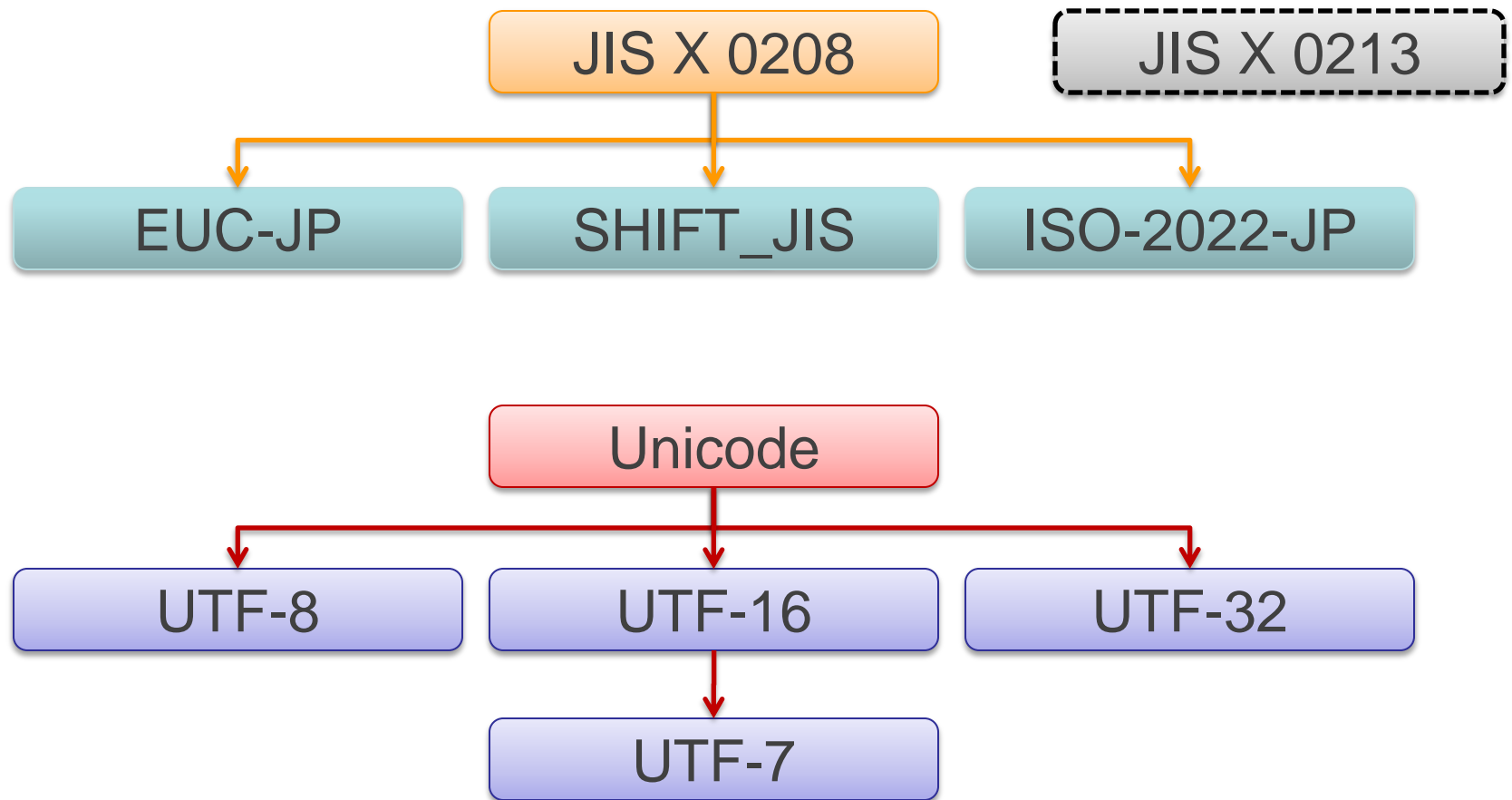
JIS X 0208
(ANSI 漢字集合)

JIS X 0201
(ANK 半角力ナ)

JIS X 0212
(補助漢字)

JIS X 0213
(ANSI 漢字集合)

文字エンコーディング



エレメントという概念 (1)

- エレメント
 - 文字の最小構成要素を指す
 - Unicode 用語では “Code Unit” と呼ばれる
- エレメントサイズ (Unicode: Code unit size)
 - ANSI のエレメントサイズは 8bit (1byte)
 - UTF-16 のエレメントサイズは 16bit (1 WORD)
- エレメント数 (Unicode: Code units)
 - Shift_JIS は、最大 2 エレメントで構成される
 - UTF-32 は必ず 1 エレメント
 - UTF-16 は最大 2 エレメント
 - UTF-8 は最大 4 エレメント

エレメントという概念 (2)

- エレメントインデックス
 - 以下のようなコードがある場合、

```
var
  A: AnsiString;
  U: UnicodeString;
  U4: UCS4String;
begin
  A := 'ABC';
  U := 'ABC';
  U4 := UnicodeStringToUCS4String('ABC');
end;
```

- 変数 A の文字 'A' のエレメントインデックスは 1、変数 U の文字 'A' のエレメントインデックスも 1、変数 U4 の文字 'A' のエレメントインデックスは 0 となる。
- “エレメント” という言葉を借りないと...
 - “バイトインデックス / ワードインデックス” と使い分けなくてはならない。
もちろん、“文字インデックス” とは違う。

ANSI – Shift_JIS

- Shift_JIS

- Windows では CP932
- 1 エLEMENTは 1 バイト
- 漢字は 2 バイト (2 エLEMENT で構成される)
- 亜種が多い。所謂、“機種依存文字” が多い。

| 名称 | 備 考 |
|----------------|---------------------|
| Shift_JIS | JIS X 0208 ベース |
| CP932 | JIS X 0208 ベース |
| Shift_JISX0213 | JIS X 0213:2000 ベース |
| Shift_JIS-2004 | JIS X 0213:2004 ベース |

他多数。

- "補助漢字 (JIS X 0212)" は使えない。
- Shift_JISX0213 と Shift_JIS-2004 は第3/4水準漢字が使える。
- 0x5C (パズデリミタ) が 2 バイト文字に含まれる場合がある。
- Delphi 1 ~ Delphi 2007 でのデフォルト文字コード (於日本)

ANSI – Shift_JIS

- 機種依存文字
 - 例えば、Mac OS X。

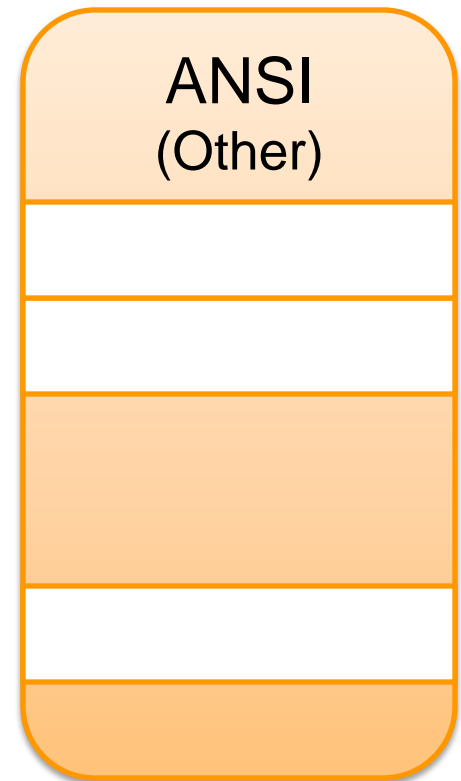
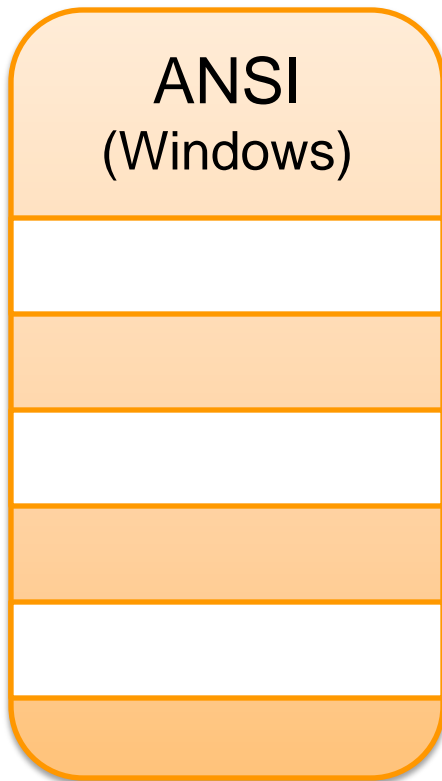
| ANSI | Windows | Mac |
|------|---------|-----|
| 0xFD | (未使用) | © |
| 0xFE | (未使用) | ™ |

0xFD と 0xFE を Unicode に変換してみると...

| ANSI | Windows | Mac |
|------|------------------|-------------|
| 0xFD | (私用領域) U+F8F1 | © U+00A9 |
| 0xFE | (私用領域) U+F8F2 | ™ U+2122 |

機種依存文字

- 機種依存文字



ANSI – Shift_JIS

- パスデリミタの問題

- 2バイト文字が考慮されていないと...

C:¥集計表1.txt

“表” は 0x95, 0x5C で、2バイト目に 0x5C (パスデリミタ) を含んでいる。
海外製のアプリケーションで問題になる事が多い。

C:¥饅頭怖い.txt

等も同様。

- コンソールアプリケーションの場合には、0x7C (パイプ) にも注意が必要。

C:¥竹の弓.txt

現在では問題になる事はあまりないとは思いますが念のため。

ANSI – Shift_JIS

- パスデリミタの問題
 - パスデリミタ (¥ = 0x5C) を含む漢字は...

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 一 | ソ | ㄣ | ㄩ | 噂 | 湮 | 欺 | 圭 | 構 | 蚕 |
| 十 | 申 | 曾 | 箎 | 貼 | 能 | 表 | 暴 | 予 | 禄 |
| 兔 | 喀 | 媾 | 彌 | 拿 | 朽 | 畎 | 濬 | 畚 | 秉 |
| 綵 | 臀 | 藹 | 觸 | 體 | 鐔 | 饅 | 鵠 | 倭 | 砒 |

- パイプ (| = 0x7C) を含む漢字は...

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 一 | 𐀃 | ㄥ | 榎 | 掛 | 弓 | 芸 | 鋼 | 旨 | 楯 |
| 酢 | 掃 | 竹 | 倒 | 培 | 怖 | 翻 | 慾 | 處 | 嘶 |
| 孝 | 忿 | 掟 | 桤 | 毫 | 烟 | 痞 | 窩 | 縹 | 艘 |
| 蛞 | 諫 | 轎 | 閑 | 驂 | 鯨 | 竣 | 蒴 | | |

ANSI – ISO-2022-JP

- ISO-2022-JP

- Windows では CP50220 / CP50221 / CP50222
- 1 エLEMENTは 1 バイト
- 漢字は 2 バイト
- 文字種はエスケープシーケンスで切り替える。
- 亜種がある。

| 名称 | 備 考 |
|------------------|---|
| ISO-2022-JP | JIS X 0208 ベース。 |
| ISO-2022-JP-1 | “半角カナ (JIS X 0201)” 利用可能 |
| ISO-2022-JP-2 | ISO-2022-JP-1 に加え、 ハングル/中国語/ラテン文字/ギリシャ文字等が利用可能 |
| ISO-2022-JP-3 | JIS X 0213:2000 ベース。 |
| ISO-2022-JP-2004 | JIS X 0213:2004 ベース。 |

- ISO-2022-JP では本来、“補助漢字 (JIS X 0212)” や “半角カナ (JIS X 0201)” は使えない。
- ISO-2022-JP は 7bit の文字コード

ANSI – ISO-2022-JP

- エスケープシーケンス

- ISO-2022-JP で文字種を切り替える時に使う。
ロッキングシフトで、他のエスケープシーケンスが現れるまで文字種は変わらない。

| | シーケンス | 内容 |
|---------------|--------------|--------------------|
| ISO-2022-JP | [ESC] (B | ASCII |
| | [ESC] (I | 半角カナ (JIS X 0201) |
| | [ESC] (J | ラテン文字 (JIS X 0201) |
| | [ESC] \$ @ | 漢字 (JIS C 6226) |
| | [ESC] \$ B | 漢字 (JIS X 0208) |
| ISO-2022-JP-1 | [ESC] \$ (D | 補助漢字 (JIS X 0212) |
| ISO-2022-JP-3 | [ESC] \$ (O | JIS X 0213 1面 |
| | [ESC] \$ (P | JIS X 0213 2面 |

CP50220 / CP50221 / CP50222 は ISO-2022-JP 準拠。

ANSI – ISO-2022-JP

- CP50220 と CP50221 と CP50222
 - Unicode の “A B C あいうアヰ” を変換した場合

CP50220 - 半角カナは **全角カナに変換** される

| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|--------|------|------|------|-------|------|------|------|------|------|------|------|------|-------|------|------|------|------|------|------|------|------|
| 文字 | A | B | C | [ESC] | \$ | B | あ | い | う | ア | イ | ウ | [ESC] | (| B | | | | | | |
| バイト列 | 0x41 | 0x42 | 0x43 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 | 0x25 | 0x22 | 0x25 | 0x24 | 0x25 | 0x26 | 0x1B | 0x28 | 0x42 |

CP50221 - 半角カナはエスケープシーケンスで表現される

| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
|--------|------|------|------|-------|------|------|------|------|------|-------|------|------|------|------|------|-------|------|------|------|------|------|
| 文字 | A | B | C | [ESC] | \$ | B | あ | い | う | [ESC] | (| I | ア | イ | ウ | [ESC] | (| B | | | |
| バイト列 | 0x41 | 0x42 | 0x43 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 | 0x1B | 0x28 | 0x49 | 0x31 | 0x32 | 0x33 | 0x1B | 0x28 | 0x42 |

CP50222 - 半角カナは<SI>/<SO>で表現される

| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|--------|------|------|------|-------|------|------|------|------|------|-------|------|------|------|------|------|------|-------|------|------|------|------|------|
| 文字 | A | B | C | [ESC] | \$ | B | あ | い | う | [ESC] | (| J | [SO] | ア | イ | ウ | [ESC] | (| B | | | |
| バイト列 | 0x41 | 0x42 | 0x43 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 | 0x1B | 0x28 | 0x4A | 0x0E | 0x31 | 0x32 | 0x33 | 0x1B | 0x28 | 0x42 |

ANSI – ISO-2022-JP

- ロッキングシフトの問題点
 - バイト列で文字連結を行うと意図しない文字種になる事がある。
‘あいう’ + ‘ABC’ は...

| | | | | | | | | | |
|--------|-------|------|------|------|------|------|------|------|------|
| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 文字 | [ESC] | \$ | B | あ | | い | | う | |
| バイト列 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 |

| | | | |
|--------|------|------|------|
| インデックス | 1 | 2 | 3 |
| 文字 | A | B | C |
| バイト列 | 0x41 | 0x42 | 0x43 |



不正

| | | | | | | | | | | | | |
|--------|-------|------|------|------|------|------|------|------|------|------|------|------|
| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 文字 | [ESC] | \$ | B | あ | | い | | う | | A | B | C |
| バイト列 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 | 0x41 | 0x42 | 0x43 |

正常

| | | | | | | | | | | | | | | | |
|--------|-------|------|------|------|------|------|------|------|------|-------|------|------|------|------|------|
| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 文字 | [ESC] | \$ | B | あ | | い | | う | | [ESC] | (| B | A | B | C |
| バイト列 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 | 0x1B | 0x28 | 0x42 | 0x41 | 0x42 | 0x43 |

ANSI – ISO-2022-JP

- ロッキングシフトの問題点 (2)

- 同一種のエスケープシーケンスをネストとして扱う処理系がある。

| | | | | | | | | | | | | |
|--------|-------|------|------|------|------|------|-------|------|------|------|------|------|
| | ネスト+1 | | | | | | ネスト+1 | | | | | |
| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 文字 | [ESC] | (| B | A | B | C | [ESC] | (| B | D | E | F |
| バイト列 | 0x1B | 0x28 | 0x42 | 0x41 | 0x42 | 0x43 | 0x1B | 0x28 | 0x42 | 0x44 | 0x45 | 0x46 |

| | | | | | | | | | |
|--------|-------|------|------|------|------|------|------|------|------|
| | ネスト-1 | | | | | | | | |
| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 文字 | [ESC] | \$ | B | | | | | | |
| バイト列 | 0x1B | 0x24 | 0x42 | 0x24 | 0x22 | 0x24 | 0x24 | 0x24 | 0x26 |

このまま文字連結すると 'あいう' は正しく処理されない。
(ネストが一段減っただけなので)

- 文字連結の問題を '文字列' + [ESC](B + '文字列' として安易に解決しようとするのはやめたほうがいい。

ANSI – EUC-JP

- EUC-JP
 - Windows では CP20932 / CP51932
 - 1 エLEMENTは 1 バイト
 - 漢字は 2 バイト (ちょっと語弊アリ)
 - 亜種がある。

| 名 称 | 備 考 |
|--------------|--|
| EUC-JP | JIS X 0208 ベース。 |
| CP20932 | 使えない文字が多い。 |
| CP51932 | “NEC選定IBM拡張文字” 利用可能。 TEncoding には指定不可。 |
| eucJP-ms | “補助漢字 (JIS X 0212)” 利用可能。 |
| EUC-JISX0213 | JIS X 0213:2000 ベース |
| EUC-JIS-2004 | JIS X 0213:2004 ベース。 |

- “半角カナ (JIS X 0201)” は 2 バイト (ちょっと語弊アリ) 。
- 実装によっては 3 バイト文字 (ちょっと語弊アリ) が存在する。

ANSI – EUC-JP

- EUC-JP の文字種切り替え

- ISO-2022-JP のような文字種切り替えが存在する。
シングルシフト。1 文字分だけ文字種を切り替える。

| | シーケンス | バイナリ | 内容 |
|--------|------------|------------------|-------------------|
| EUC-JP | SS2 (0x8E) | 0x8E, 0xXX | 半角カナ (JIS X 0201) |
| | SS3 (0x8F) | 0x8F, 0xXX, 0xXX | 補助漢字 (JIS X 0212) |

- シングルシフトなので...
同じ文字種が続いても、1 文字毎に必ずシフト指定をする必要がある。
半角カナ は 1 バイト文字だけれど、実質 2 バイト必要。
補助漢字は 2 バイト文字だけれど、実質 3 バイト必要。
- euc-JP-ms は補助漢字が扱えるため ” 3 バイト文字 ” が存在する。

ANSI – EUC-JP

- CP20932 と CP51932
 - Unicode の “— // — ~ ¢ £ ¬” を変換した場合

CP20932

| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|--------|------|------|------|------|------|------|------|------|------|------|
| 文字 | — | | — | | ? | ? | ? | ? | ? | ? |
| バイト列 | 0xA1 | 0xBD | 0xA1 | 0xB1 | 0x3F | 0x3F | 0x3F | 0x3F | 0x3F | 0x3F |

CP51932

| インデックス | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|--------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 文字 | — | | — | | // | | — | | ~ | | ¢ | | £ | | ¬ | |
| バイト列 | 0xA1 | 0xBD | 0xA1 | 0xB1 | 0xA1 | 0xC2 | 0xA1 | 0xDD | 0xA1 | 0xC1 | 0xA1 | 0xF1 | 0xA1 | 0xF2 | 0xA2 | 0xCC |

CP20932 では所謂 “波ダッシュ問題” が発生する (後述)。

Unicode

- Unicode は 21bit の文字集合
 - U+0000 ~ U+10FFFF の範囲。
- 16bit ...65,536 コードポイントを“プレーン”という単位で扱う
 - $0x110000 \div 0x10000 = 17$ なので、17 プレーン存在する。
 - コードポイントの上位 5bit がプレーン番号となる。
- 最初のプレーンは U+0000 ~ U+FFFF で、16bit の範囲にある
 - プレーン 0 は 基本多言語面 (Basic Multilingual Plane) と呼ばれる。
- プレーンの中にも細かい区切りがある
 - 国別 / 言語別 / 用途別 のエリアがある。

Unicode – プレーン

| | | | |
|---|--|---|---|
| Plane #0 (BMP=UCS-2) U+0000-U+FFFF | | | |
| Plane #1 (SMP) U+10000- U+1FFFF | Plane #2 (SIP) U+20000- U+2FFFF | Plane #3 (reserved) U+30000- U+3FFFF | Plane #4 (reserved) U+40000- U+4FFFF |
| Plane #5 (reserved) U+50000- U+5FFFF | Plane #6 (reserved) U+60000- U+6FFFF | Plane #7 (reserved) U+70000- U+7FFFF | Plane #8 (reserved) U+80000- U+8FFFF |
| Plane #9 (reserved) U+90000- U+9FFFF | Plane #10 (reserved) U+A0000- U+AFFFF | Plane #11 (reserved) U+B0000- U+BFFFF | Plane #12 (reserved) U+C0000- U+CFFFF |
| Plane #13 (reserved) U+D0000- U+DFFFF | Plane #14 (SSP) U+E0000- U+EFFFF | Plane #15 (PUP #A) U+F0000- U+FFFFF | Plane #16 (PUP #B) U+100000- U+10FFFF |

BMP は Plane #0

サロゲートペアに対応しない UTF-16 や、3 バイトしか使えない UTF-8 (後述) は、総コードポイントのたった **1 / 17** しか使えないことになる。

...ただ、現時点では Plane #3 ~ Plane #13 までは未定義となっている。

Unicode – UTF-8

- UTF-8
 - Windows では CP65001 (CP_UTF8)
 - 1 エLEMENTは 1 バイト
 - 1～4 バイトのマルチバイト
 - 亜種がある。

| 名称 | 備 考 |
|----------------|---|
| UTF-8 | Unicode をマルチバイトで表現したもの。 U+10000 以降の文字は 4 バイトで表現される。 |
| UTF-8N | UTF-8 の BOM 無し |
| CESU-8 | UTF-16 を UTF-8 符号化したもの。 U+10000 以降の文字は “3 バイト × 2” で表現される。 |
| UTF-FSS | BMP (UCS-2) のみの UTF-8。UTF-8 の旧名称でもある。 |
| Unicode-FSS | UTF-FSS 同様。Interbase / Firebird で用いられる。 |
| Modified UTF-8 | CESU-8 同様。NULL 文字は 0xC0, 0x80 にエンコードされる。 |

- BMP 以外をサロゲートペアと呼ぶのは間違い。UTF-8 なら “4 バイト文字”。
- セキュリティに注意。

Unicode – UTF-8

- CESU-8 と Modified UTF-8

UTF-8



CESU-8 / Modified UTF-8



- 3 バイト UTF-8 の一種。
- CESU-8 / Modified UTF-8 は 21 bit の Unicode Codepoint に変換せず、UTF-16 のエレメント (16bit) のまま UTF-8 エンコードしている。
- サロゲートペアは 3 バイト文字 × 2 で表現される。
- 通常の UTF-8 だと思って読み込むと不正な文字列となる。

Unicode – UTF-8

- UTF-8 の構造
 - 1 ～ 4 バイト文字の構造

U+0000～U+007F (BMP)

| | | | | | | | | |
|-----|---|------|---|---|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 0 | 7ビット | | | | | | |

U+0080～U+07FF (BMP)

| | | | | | | | | |
|-----|---|---|---|--------|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 0 | 上位5ビット | | | | |

| | | | | | | | | |
|-----|---|---|--------|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 下位6ビット | | | | | |

U+0800～U+FFFF (BMP)

| | | | | | | | | |
|-----|---|---|---|---|--------|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 1 | 0 | 上位4ビット | | | |

| | | | | | | | | |
|-----|---|---|------|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 6ビット | | | | | |

| | | | | | | | | |
|-----|---|---|--------|---|---|---|---|---|
| 3rd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 下位6ビット | | | | | |

U+10000～U+10FFFF

| | | | | | | | | |
|-----|---|---|---|---|---|--------|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 1 | 1 | 0 | 上位3ビット | | |

| | | | | | | | | |
|-----|---|---|------|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 6ビット | | | | | |

| | | | | | | | | |
|-----|---|---|------|---|---|---|---|---|
| 3rd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 6ビット | | | | | |

| | | | | | | | | |
|-----|---|---|--------|---|---|---|---|---|
| 4th | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 下位6ビット | | | | | |

- 8bit 目が 0 なら 1 バイト文字
- 8bit 目が 1 で 7bit 目が 0 ならマルチバイト文字の 2 文字目以降
- 7 / 8bit 目が 1 で 6bit 目が 0 なら 2 バイト文字の先頭
- 7 / 8bit 目が 1 で 5bit 目が 0 なら 3 バイト文字の先頭
- 7 / 8bit 目が 1 で 4bit 目が 0 なら 4 バイト文字の先頭

Unicode – UTF-8

- UTF-8 のセキュリティ問題
 - 同じコードポイントを別のバイト列に符号化できてしまう。

0x5C (正常)

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |

0xC1, 0x9C (不正)

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

0xE0, 0x81, 0x9C (不正)

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 3rd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

0xF0, 0x80, 0x81, 0x9C (不正)

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 1st | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 2nd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 3rd | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| 4th | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |

- これらを不用意に変換するとすべて 0x5C を表す。
- パスデリミタやドットを不正な符号化で表現すればディレクトリトラバーサルが行える (..¥..¥ とか)。
- 不正な符号化は 0x5C 等の検索をすり抜ける場合がある。
- 自前での変換には十分な注意が必要となる。

Unicode – UTF-16

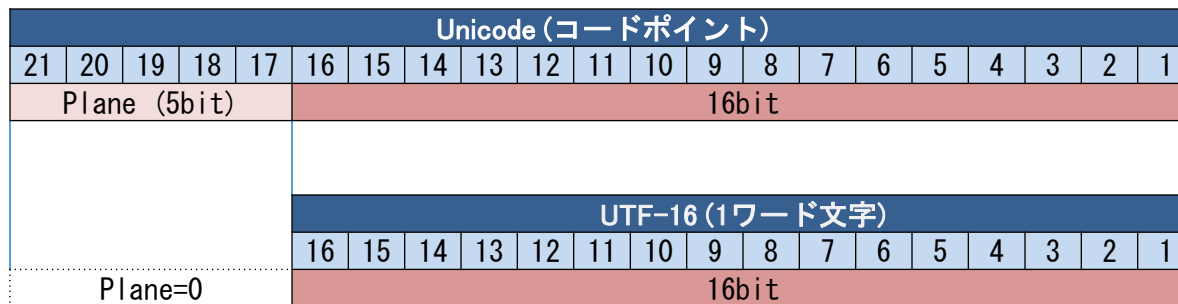
- UTF-16
 - Windows では CP1200
 - 1 エLEMENTは 1 ワード
 - 1～2 ワードのマルチワード
 - 基本多言語面 (BMP) 以外の文字はサロゲートペア (2 ワード) となる
 - 1ワードのみで構成される UTF-16 は UCS-2 と同等。
 - 亜種は....

| 名称 | 備 考 |
|-------------|---|
| UTF-16 (LE) | Unicode をマルチワードで表現したもの。 U+10000 以降の文字はサロゲートペアで表現される。 |
| UTF-16BE | UTF-16 の エンディアンがビッグエンディアンになったもの。 |
| UCS-2 | 1 ワードしか使わない UTF-16 と同義。 U+10000 以降の文字は使えない。 |

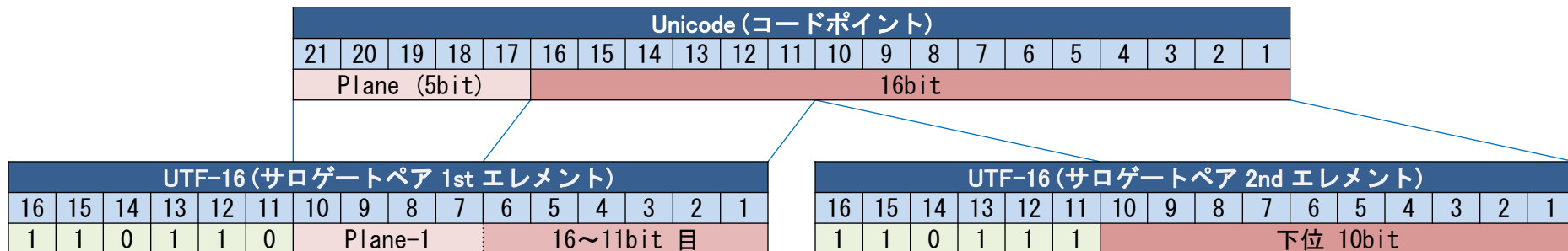
- Delphi 2009 以降でのデフォルト文字コード

Unicode – UTF-16

- UTF-16 の構造
 - 1 ワード文字の構造



- 2 ワード文字 (サロゲートペア) の構造



Unicode – その他の UTF

- UTF-32

- Windows では CP12000
- 1 エLEMENTは 1 ダブルワード
- 主に内部でコードポイント単位の処理をする際に使う。
- ファイルとして扱うことはまずない。
- UCS-4 と同等。
- TEncoding では扱えない。

- UTF-7

- Windows では CP65000 (CP_UTF7)
- 電子メール等、7 bit しか処理できない処理系向け。
- 実は UTF-16 を BASE64 化したもの (ざっくり言えば)。
- ASCII との区別が付かない。
- セキュリティに注意。

Unicode – UTF との関係

- Unicode と UTF まとめ

| | Code Point | | | |
|-------------------------|------------------------|------------------------|------------------------|------------------------------------|
| | U+0000～U+007F (BMP) | U+0080～U+07FF (BMP) | U+0800～U+FFFF (BMP) | U+10000～U+10FFFF (Plane #1～#16) |
| Code Point 数 | 128 | 1,920 | 63,488 | 1,048,576 |
| Code Point 比 (BMP 比) | 0.01% (0.20%) | 0.17% (2.93%) | 5.70% (96.88%) | 94.12% (---) |
| 文字割当 | ASCII | ギリシャ文字等 | 漢字等 | 追加漢字等 |
| UTF-8 | 1 byte | 2 byte | 3 byte | 4 byte |
| UTF-16 | 1 WORD | | | 2 WORD (サロゲートペア) |
| UTF-32 | 1 DWORD | | | |

Unicode – 正規化 (1)

- 正規化とは？
 - 比較のために片方、または両方を同じルールに従って変換する事を言う
 - If (AnsiUpperCase(S1) = AnsiUpperCase(S2)) then
文字列変数を大文字化して比較...これも正規化
 - 名寄せ...これも正規化
 - カタカナの“アイウエオヤユヨツ”を“アイウエオヤヨツ”に変換
 - 高橋さんと 高(はしごだか)橋さんを同じに扱う
- 何故正規化しなくてはならないのか？
 - 見た目が同じなのに比較で一致しない
 - DB とかだと Where 句の条件に一致しないので、
全レコードをスキャンするハメになる。
- では“Unicode の正規化”とは...？

Unicode – 正規化 (2)

- 合成済み文字と結合文字列
 - 1 コードポイントで表される 合成済み文字
が = U+304C
 - 複数のコードポイントが連結された結合文字列
が = か(U+304B) + ` (U+3099)
 - 当然、単純比較では一致しない
 - Mac OS X では結合文字列と合成済み文字が存在する場合にはデフォルトで結合文字列が使われる (ファイル名とか)
- 互換文字
 - “使わない事が推奨されている” 文字
 - “率” のコードポイントは U+7387
 - 互換文字の “率” は U+F961 の他に U+F9DB がある。
 - ダイアクリティカルマーク付の合成済み文字 (結合文字列 ではない) も互換文字

Unicode – 正規化 (3)

- Unicode 正規化の種類
 - NFD (Normalization Form Canonical Decomposition)
正順等価性で分解する。合成済み文字は結合文字列になる。
Mac OS X のファイル名は、この NFD で正規化されている。
 - NFC (Normalization Form Canonical Composition)
正順等価性で分解し、正順等価性で合成する。
結合文字列は合成済み文字になる。
 - NFKD (Normalization Form Compatibility Decomposition)
互換等価性で分解する。合成済み文字は結合文字列になる。
 - NFKC (Normalization Form Compatibility Composition)
互換等価性で分解し、正順等価性で合成する。
結合文字列は合成済み文字になる。

Unicode – 正規化 (4)

- 試しに “カ” (U+FF76 , U+FF9E) を正規化してみる
 - NFD: “カ” (U+FF76 , U+FF9E)
半角のまま
 - NFC: “カ” (U+FF76 , U+FF9E)
合成済み文字列の 半角 “カ” はないので半角のまま
 - NFKD: “カ” (U+30AB + U+3099)
半角 “カ” は全角の結合文字列になる。
 - NFKC: “カ” (U+30AC)
半角 “カ” は全角の合成済み文字になる。

Unicode – 正規化 (5)

- 互換漢字はどうなるの？
 - 互換漢字の“**率** (U+F961 or U+F9DB)”は“**率** (U+7387)”へ正規化される
 - 互換漢字の“**崎** (たちぎき: U+FA11)”は“**崎** (U+ 5D0E)”へ正規化されない。
 - 互換漢字は統合漢字へ正規化されたりされなかったりする。
 - 少なくとも、
見た目が全く同じ文字はどの正規化方法を用いても統合漢字へ正規化される。
- 結局、どれを使えばいいの？
 - 日本語だけに限って言えば、NFC か NFKC。
理由としては“日本語 ANSI では結合文字列が使われていなかった”から。

Unicode – 異体字

- 異体字
 - ‘**広**’の旧字体である‘**廣**’には、異体字がある。

廣 廣

実はどちらもコードポイントは U+5EE3 で、違うのはフォントだけ(右側は SimHei)

- 異体字セレクトは
U+5EE3 , **U+E0100** のように使う。ここが **U+E0101** になると、先行するコードポイントに対して別のフォントが選択され、正しく異体字が表示されるという仕組みになっている。
- ちなみに、Unicode の例字では右側の‘廣’となっている。
実は日本語の方がマイノリティ。

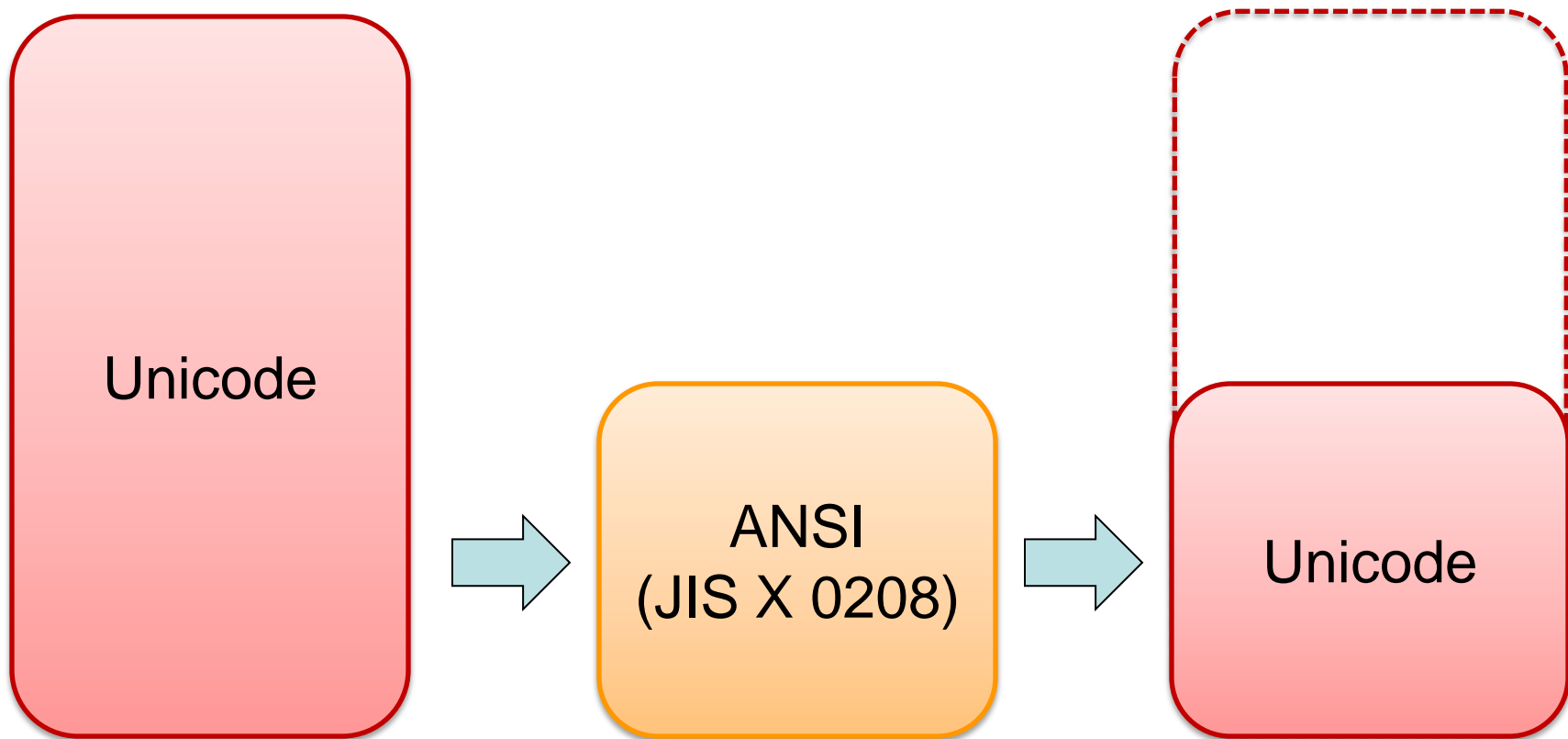
文字コード – ラウンドトリップ

- ラウンドトリップ

- 文字描画幅を得るために Unicode → ANSI → Unicode 変換を安易に行ってはいけない。ANSI に存在しない文字が失われる。
- 文字単位での処理を行うために ANSI → Unicode → ANSI 変換を行ってはいけない。
例えば、Shift_JIS の 0x81, 0xE6, 0x87, 0x9A, 0xFA, 0x5B は “.: :. :.” となるが、これを Unicode へ変換して Shift_JIS へ戻すと 0x81, 0xE6, 0x81, 0xE6, 0x81, 0xE6 となり、“**字形が同じな別の文字**” は同一のコードポイントにまとめられてしまう。

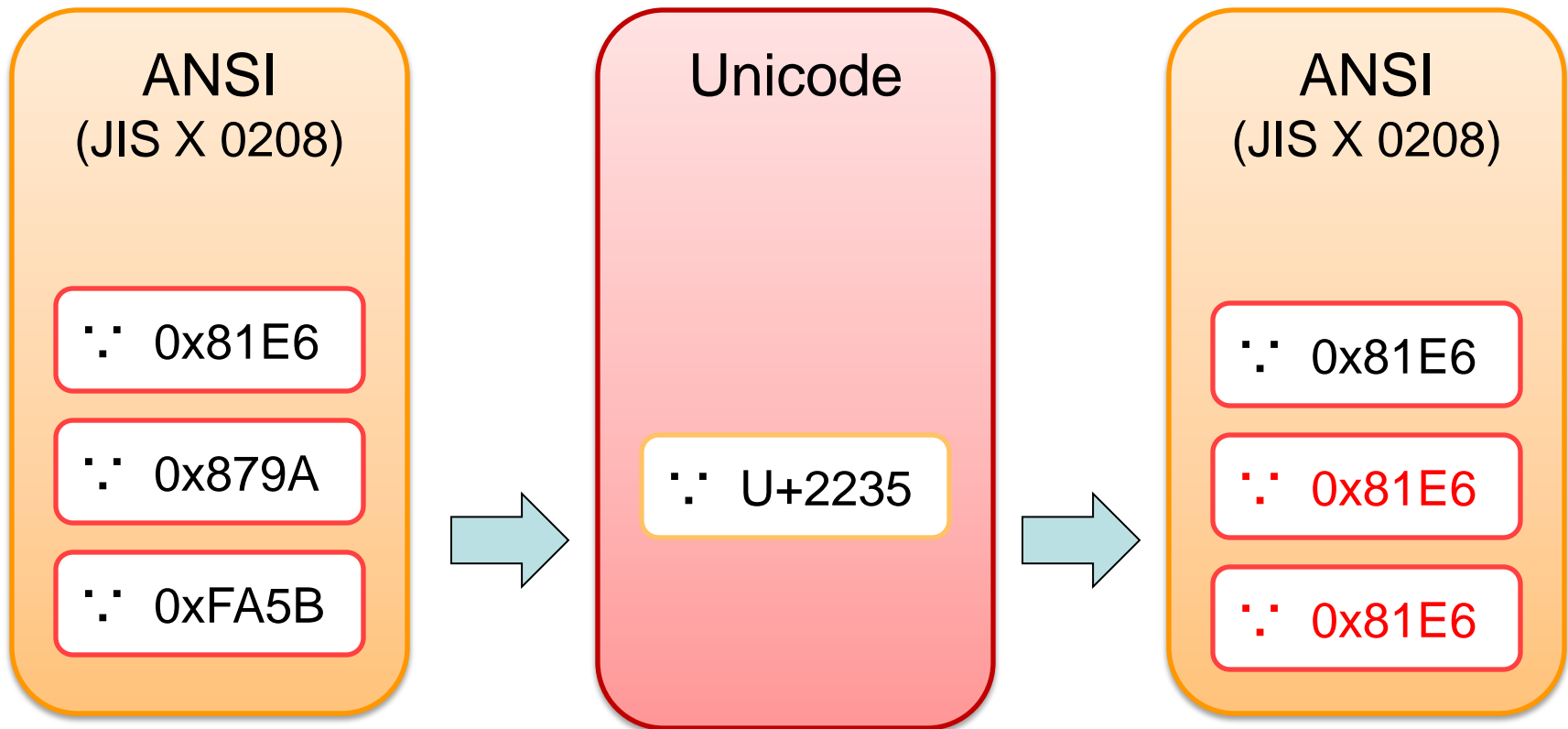
ラウンドトリップ (1)

- Unicode -> ANSI -> Unicode



ラウンドトリップ (2)

- ANSI -> Unicode -> ANSI



文字コード – マッピング

– Unicode から ANSI へのマッピング (Windows)

| Unicode | | | Shift_JIS | ISO-2022-JP | | | EUC-JP | |
|---------|-----------|------------------------|-----------|-------------|---------|---------|---------|---------|
| Glyph | CodePoint | Name | CP932 | CP50220 | CP50221 | CP50222 | CP20932 | CP51932 |
| ¢ | U+00A2 | CENT SIGN | U+FFE0 | U+FFE0 | U+FFE0 | U+FFE0 | U+FFE0 | U+FFE0 |
| ¢ | U+FFE0 | FULLWIDTH CENT SIGN | | × | × | × | × | |
| £ | U+00A3 | POUND SIGN | U+FFE1 | U+FFE1 | U+FFE1 | U+FFE1 | U+FFE1 | U+FFE1 |
| £ | U+FFE1 | FULLWIDTH POUND SIGN | | × | × | × | × | |
| ¬ | U+00AC | NOT SIGN | U+FFE2 | U+FFE2 | U+FFE2 | U+FFE2 | U+FFE2 | U+FFE2 |
| ¬ | U+FFE2 | FULLWIDTH NOT SIGN | | × | × | × | × | |
| – | U+2014 | EM DASH | × | × | × | × | × | × |
| — | U+2015 | HORIZONTAL BAR | | | | | | |
| | U+2016 | DOUBLE VERTICAL LINE | × | U+2225 | U+2225 | U+2225 | U+2225 | × |
| // | U+2225 | PARALLEL TO | | × | × | × | × | |
| - | U+203E | OVERLINE | × | × | × | × | × | × |
| — | U+FFE3 | FULLWIDTH MACRON | | | | | | |
| – | U+2212 | MINUS SIGN | × | U+FF0D | U+FF0D | U+FF0D | U+FF0D | × |
| — | U+FF0D | FULLWIDTH HYPHEN-MINUS | | × | × | × | × | |
| ~ | U+301C | WAVE DASH | × | U+FF5E | U+FF5E | U+FF5E | U+FF5E | × |
| ~ | U+FF5E | FULLWIDTH TILDE | | × | × | × | × | |

JIS X 0208 には存在しない

青文字または × の文字は Unicode → ANSI → Unicode のラウンドトリップ不可

文字コード – マッピング

– ANSI から Unicode へのマッピング (Windows)

| ANSI | | Unicode | | | | | | | |
|-------------|----------|---------|--------|--------|--------|--------|--------|--------|--------|
| Name | CodePage | ¢ | £ | ¬ | — | // | — | — | ~ |
| | | U+FFE0 | U+FFE1 | U+FFE2 | U+2015 | U+2225 | U+FFE3 | U+FF0D | U+FF5E |
| Shift_JIS | CP932 | | | | | | | | |
| ISO-2022-JP | CP50220 | U+00A2 | U+00A3 | U+00AC | | U+2016 | | U+2212 | U+301C |
| | CP50221 | U+00A2 | U+00A3 | U+00AC | | U+2016 | | U+2212 | U+301C |
| | CP50222 | U+00A2 | U+00A3 | U+00AC | | U+2016 | | U+2212 | U+301C |
| EUC-JP | CP20932 | U+00A2 | U+00A3 | U+00AC | | U+2016 | | U+2212 | U+301C |
| | CP51932 | | | | | | | | |

| | |
|--|--------------------------|
| | JIS X 0208 には存在しない |
| | 場合によってはU+2014 (EM DASH) |
| | 場合によってはU+203E (OVERLINE) |

青文字は ANSI → Unicode → ANSI のラウンドトリップ不可

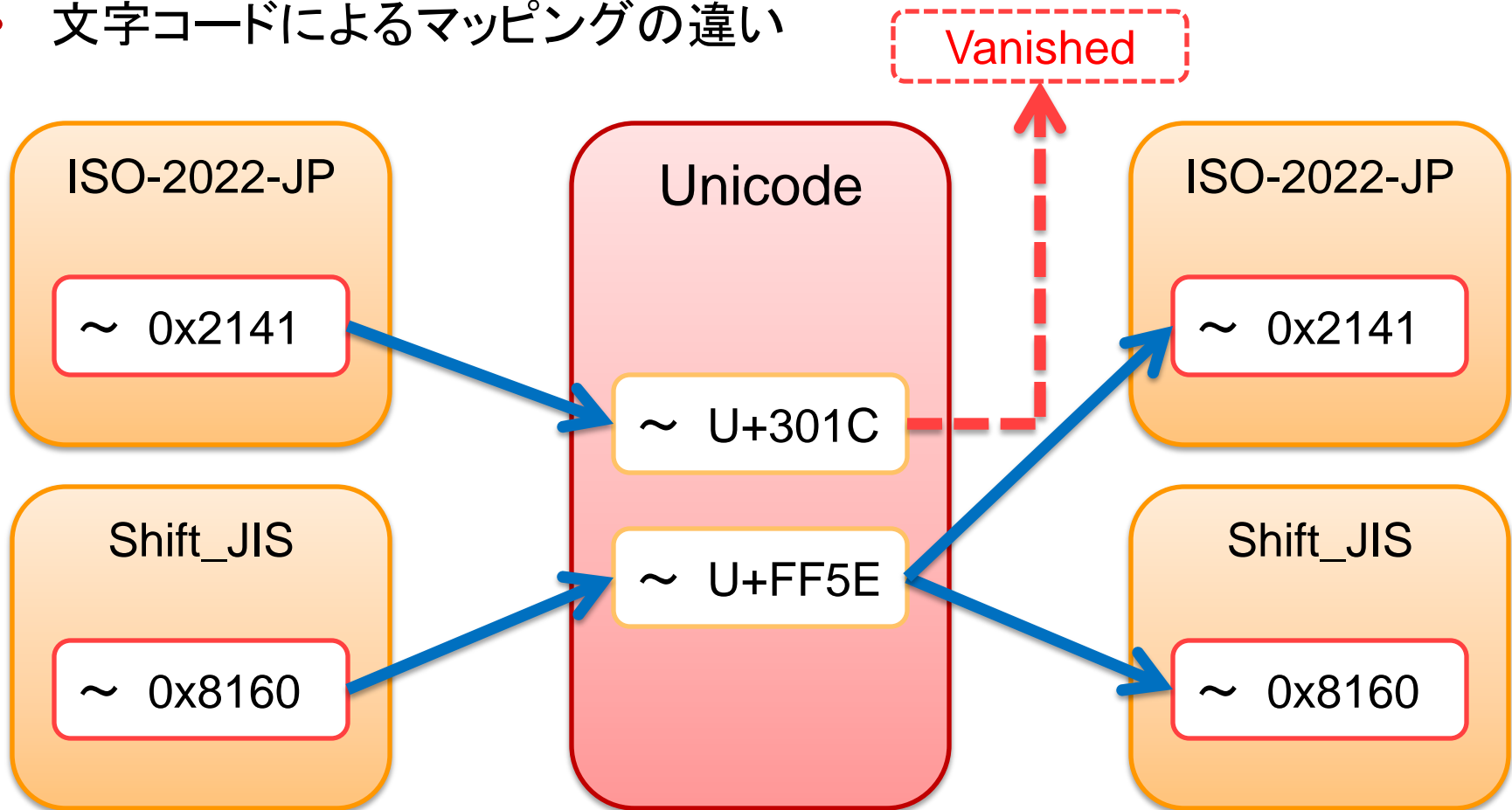
文字コード – マッピング

- 文字コードマッピング

- ANSI からラウンドトリップすると文字が正しく表示されない事がある。
これは“全角チルダ / 波ダッシュ問題“と呼ばれている。
- Unicode からラウンドトリップすると文字が正しく表示されなかったり、別の文字に置換される事がある。
- 機種依存文字とは別に、ANSI から Unicode、
または Unicode から ANSI へ変換すると文字が正しく表示されない事がある。
Unicode (Windows) → Shift_JIS (Mac) や Shift_JIS (Mac) → Unicode (Windows) またはその逆の場合である。
- OS によって ANSI ⇔ Unicode のマッピングに違いがある。
ANSI ⇔ Unicode のマッピング方法は 1 種類だけではない。

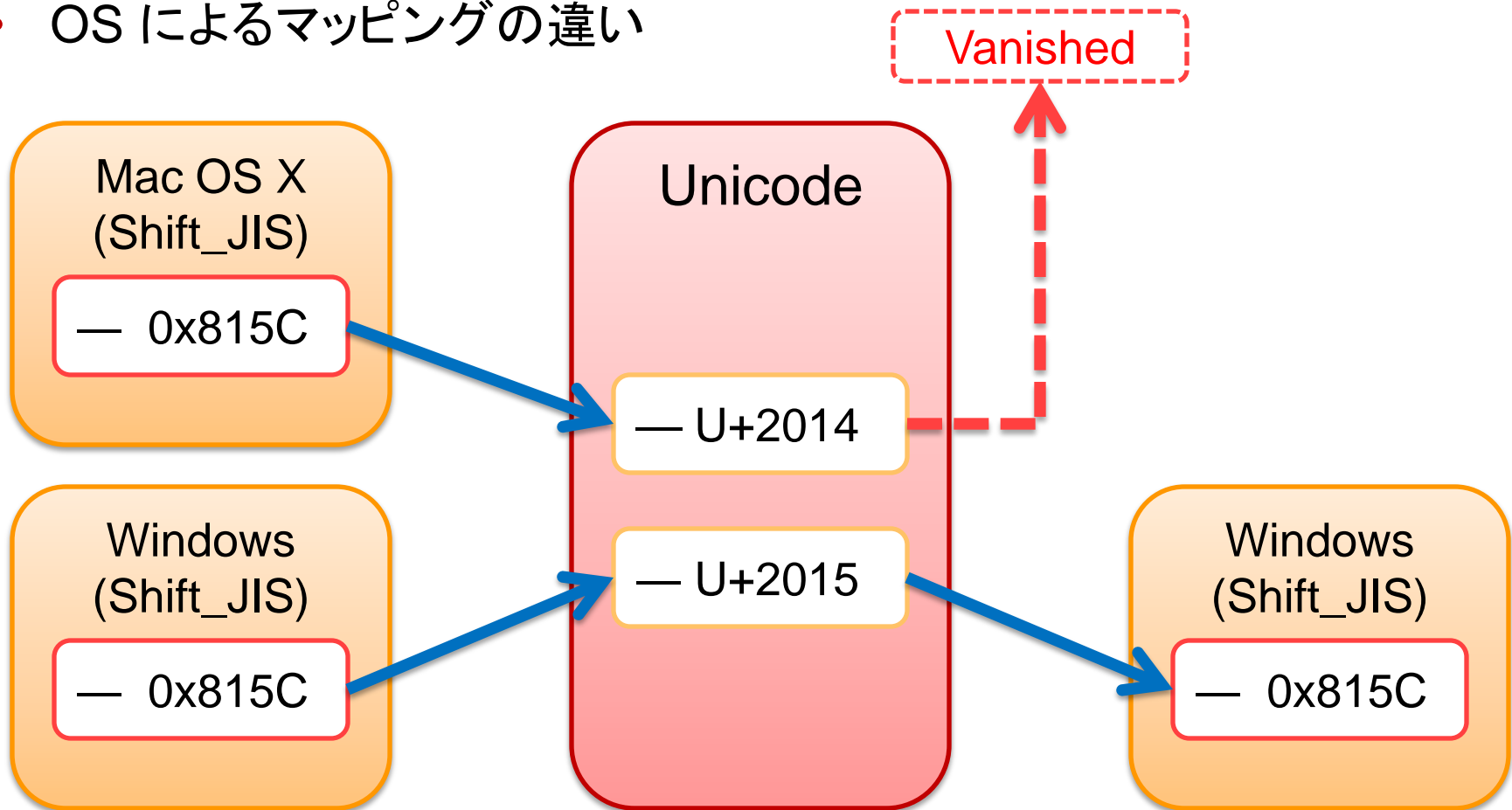
マッピング (1)

- 文字コードによるマッピングの違い



マッピング (2)

- OS によるマッピングの違い



文字コード – マッピング

- 新常用漢字問題

- 新常用漢字には、JIS X 0208 に含まれない文字がある。
- これらの文字は普通の Shift_JIS / ISO-2022-JP / EUC-JP では扱えない。

| JIS X 0213 | JIS X 0208 互換文字 |
|-------------|-----------------|
| 剥 (U+525D) | 剥 (U+5265) |
| 填 (U+5861) | 填 (U+586B) |
| 頬 (U+9830) | 頬 (U+982C) |
| 叱 (U+20B9F) | 叱 (U+53F1) |

特に‘叱’はサロゲートペアとなっている。

教育機関等では、このたった一文字のためにサロゲートペアへの対応をしなくてはならない事になる。字形が似ているので JIS X 0208 のもので代用してもいいような気がするが、書き順が違うという問題がある。

文字化け

- 機種依存文字
 - OSの差異による収録文字の違い
 - 機種の差異による収録文字の違い
- 文字コード依存
 - 文字コードの違いによる収録文字の違い
- 文字集合依存
 - 文字コードは同じだが、文字集合による収録文字の違い
- 文字コードマッピング依存1 (⇔ Unicode)
 - OS が同じで文字集合も同じだが、文字コードによりマッピング方法が異なる
- 文字コードマッピング依存2 (⇔ Unicode)
 - 文字コードは同じで文字集合も同じだが、OS 等によりマッピング方法が異なる
(ルールは一応存在するが、ベンダ独自のマッピングになっている！！)
- 複合的な要因がある場合には**対処療法では絶対に上手くいかない。**

文字化け

- ANSI は他の ANSI へ変換しても、Unicode へ変換しても文字化けの可能性が常に付きまとう
 - 正直やってられません
- UTF はどの形式でも、コードポイントは同一
 - Unicode は、どの UTF に変換してもロスレス変換となる
 - OS / 機種によってはフォントが存在しないという事はある
 - 当然ながら私的利用領域 (いわゆる外字エリア) の文字は意図した文字で表示できない事がある。
(U+E000～U+F8FF 及び、Plane #15 / #16)
 - ちなみに、Mac OSX で U+F8FF は Apple ロゴ

文字幅

- 固定ピッチフォント

- 以下はフォントサイズが同じで、どちらも固定ピッチフォント。

MS ゴシック: あいう A B Γ Δ Y Z H Θ I 123

Tahoma: あいう ABΓΔYZHΘI123

ギリシャ文字は Shift_JIS では 2 バイト文字だが、いわゆる半角で描画される。

- “バイト数＝文字幅” という考えはやめた方がいい。

文字幅

- Unicode と文字幅
 - Unicode では、コードポイント毎に文字幅の属性が設定されている。
 - Fullwidth : 全角 (F)
 - Halfwidth : 半角 (H)
 - Wide : 広い (W)
 - Narrow : 狭い (Na)
 - Ambiguous : 曖昧 (A)
 - Neutral : 中立 (N)
 - この文字幅に関する参考規定を East Asian Width (EAW) という。
<http://ja.wikipedia.org/wiki/東アジアの文字幅>
 - 最新の属性テーブルはこちら
<http://www.unicode.org/Public/UNIDATA/EastAsianWidth.txt>

文字幅

- EAW と固定ピッチの関係
 - 固定ピッチでは“半角：全角 = 1：2”の比率。
 - しかし、上記条件では Neutral 属性の文字は使えない

Neutral 属性の文字には“Æ” (U+00C6) のような文字があり、これを半角とみなすとなると、全角は...

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | Æ | あ | い | う |
|---|---|---|---|---|---|---|

こうするしかなくなってしまう。では、Neutral 属性 を全角にすると？

| | | | | | | |
|---|---|---|---|---|---|---|
| T | i | b | u | r | ó | n |
|---|---|---|---|---|---|---|

文字幅

- EAW と固定ピッチの関係
 - 日本語では Ambiguous 属性の文字はいわゆる全角として扱う。
 - 先述のギリシャ文字は Ambiguous 属性。
東アジアではいわゆる全角、東アジア以外ではいわゆる半角で扱う。
よって、“東アジアの判定”を何らかの方法で行う必要がある。
 - Fullwidth / Wide は全角サイズ、Halfwidth / Narrow は半角サイズで扱う。
 - Neutral 属性の文字を使わず、Ambiguous 属性を適切に処理すれば
Unicode においても、“半角：全角 = 1：2”を成立させる事が可能。
 - 言うまでもなく、
Unicode ではエレメント数やコードポイント数から文字幅を算出する事は不可能。

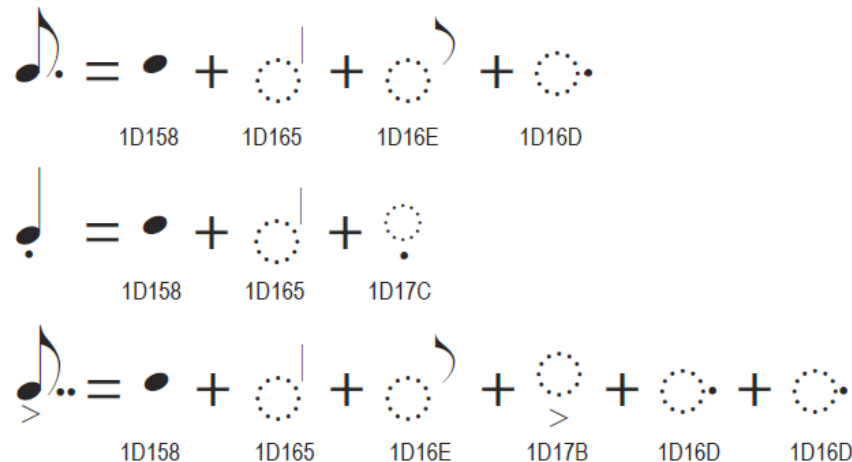
結合文字列の文字幅

- 結合文字列

- 以下の結合文字列は理論上、1 文字分の幅となるはず。

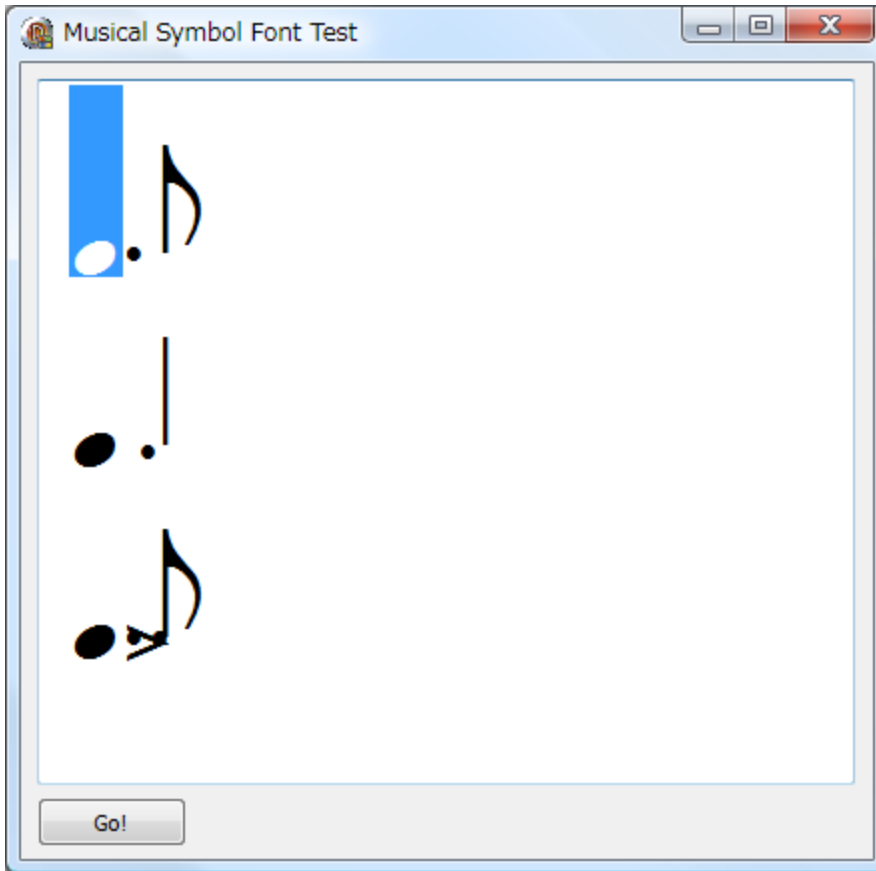
<http://www.unicode.org/versions/Unicode5.0.0/ch15.pdf>

Figure 15-11. Augmentation Dots and Articulation Symbols



結合文字列の文字幅

- これが現実。



つまり、文字幅の計算を
コードポイントから行うのは

不可能！！

キャレットの状態を見ての通り、
正常に結合されていない。

結合文字列を考慮して、
固定幅で1文字ずつ描画すると
いう事も不可能に近い。

描画してみない事には文字幅は
ワカラナイのだから。

Windows – フォント

- Unicode（日本語）が使えるフォント
 - メイリオ (プロポーショナル)
Windows XP 以降で利用可能。JIS X 0213 : 2004 の文字をすべて収録。
 - IPA フォント (プロポーショナル / 固定ピッチ)
<http://ossipedia.ipa.go.jp/ipafont/> JIS X 0213 : 2004 の文字をすべて収録。
 - 和田研細丸ゴシック2004 (プロポーショナル/ 固定ピッチ)
<http://sourceforge.jp/projects/jis2004/wiki/>
JIS X 0213 : 2004 の文字をすべて収録。ARIB 文字を収録したものもある。
 - Arial Unicode MS (プロポーショナル)
Microsoft Office に収録されている。Unicode 2.1 規格のすべての文字を網羅。
 - MingLiU / SimSun(プロポーショナル/ 固定ピッチ)
中国語フォントだが、日本語文字を含んでいる。

Windows – バグ他

- Windows 2000 ではデフォルトでサロゲートペアが扱えない
 - [HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Windows NT¥CurrentVersion¥LanguagePack] の “SURROGATE” を 0x00000002 (REG_DWORD) に設定。
東アジア版以外の Windows XP / Server 2003 も同様。
- 結合文字列の処理は OS のバージョンに依存する
 - メモ帳に”**が** (U+304B,U+3099)” をペーストして BackSpace を押すと？
- 結合文字列の処理はフォントに依存する
 - フォントによっては ”**が** (U+304B,U+3099)” は “**か**” と 2 文字になる。
- Windows は文字集合としての JIS X 0213 をサポートする
 - 但し、JIS X 0213 を文字集合として利用する文字エンコーディングには対応していない (Ex. Shift_JIS-2004)

Windows – バグ他

- Windows 2000 では WideCharToMultibyte() がバッファの計算を間違う。
 - 内部で WideCharToMultibyte() を使っている VCL/RTL も影響を受ける。
例えば、TEncoding で CP50220 / CP50221 / CP50222 へ変換すると文字列が切り詰められる事がある。Indy でも同様な現象が発生する。
- OS によって、ISO-2022-JP のエスケープシーケンスの挙動が違う
 - 変換後の末尾に **[ESC](B** が付いたり付かなかったりする。
- 異体字セレクタは Windows 7 + 対応フォントでのみ動作する。
 - デフォルト状態の日本語版 Windows 7 では動作しない。
 - 現状では深く考えない方がいい。

雑多な情報

- Windows で文字描画を細かく制御したい場合には Uniscribe を用いる。
<http://msdn.microsoft.com/ja-jp/library/cc422022.aspx>
- Windows での Unicode のグリフ描画は USP10.dll のバージョンに左右される。
- “SBCS は 1 バイト= 1 文字”...ではない。
タイ語は ANSI (CP874) でも Unicode でも結合文字列で表現される。
(Ex. สวัสดี : 4 文字、6 bytes / Code point)
- “SBCS はバイト単位で文字を切り出せる” というのは正しくない。



本題。

まず...

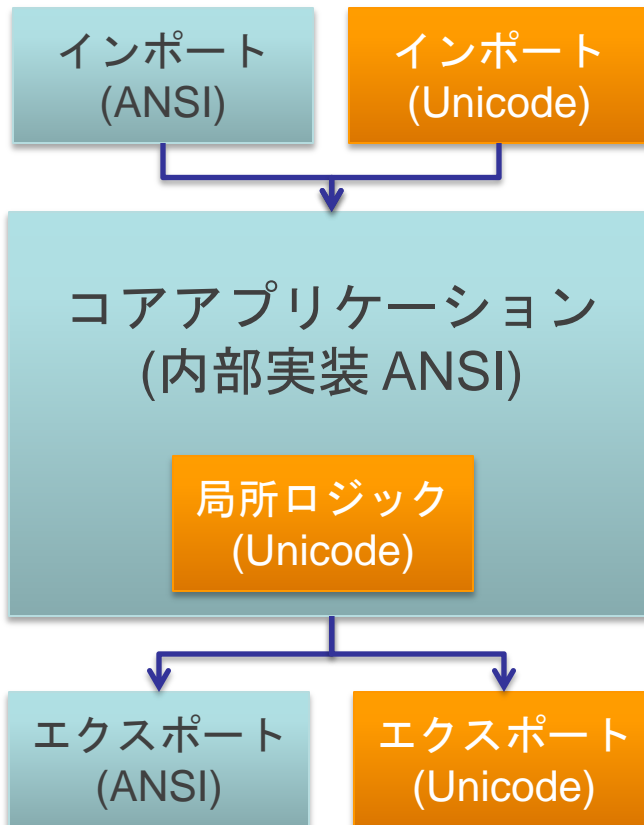
- 先程までの話は、Delphi に限った事ではない。
- 文字コードに対する基礎知識があれば、複雑なコーディングをする場面は限られてくる。
- 少なくとも、Unicode ベースのアプリケーションは、トランスレーションだけ行えば海外でも使える。
 - フォント変更の GUI は必須となる。
 - 文字方向が異なる場合は例外。この場合、設計を UI から見直す必要がある。
- ここから先は Delphi に関連した話。
 - “ANSI 版 Delphi” とは Delphi ~ Delphi 2007 を指す。
 - “Unicode 版 Delphi” とは Delphi 2009 / 2010 を指す。

方向性

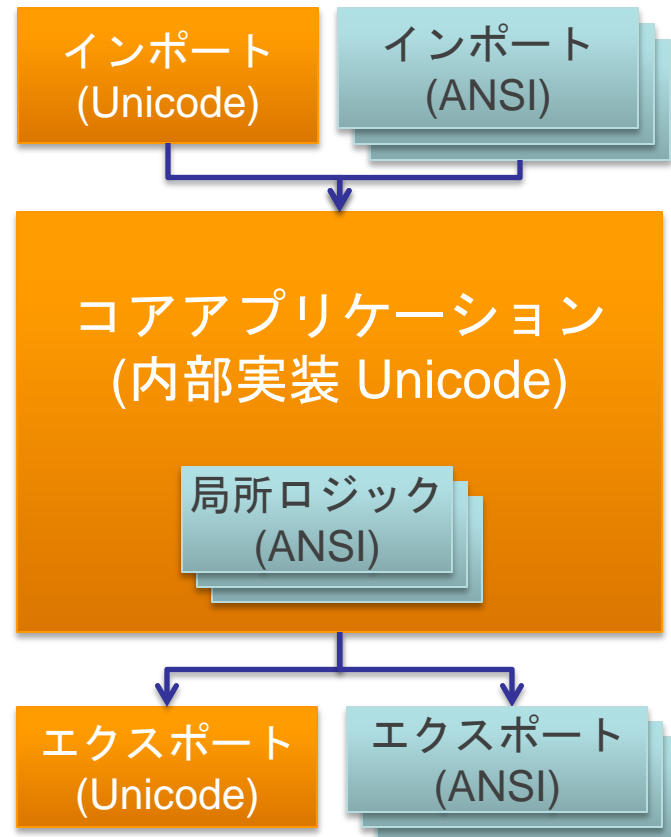
- 基本的に ANSI 版 Delphi では Unicode をファイル入出力以外では使わない
 - VCL / RTLが ANSI を前提としている
- 基本的に Unicode 版 Delphi では ANSI 文字コードをファイル入出力以外では使わない
 - VCL / RTLが UTF-16 を前提としている
- ANSI / Unicode 両用のアプリケーションはできるだけ作らない
 - `{$IFDEF Unicode}` での分岐が多くなりすぎる。
 - ソースコードの可読性が下がる。
 - Unicode と ANSI はそれぞれ別の問題点と解決方法がある。

方向性

ANSI 版 Delphi アプリケーション



Unicode 版 Delphi アプリケーション



ファイル入出力

- ANSI 版 Delphi で、Unicode 形式ファイルを扱う場合には WideStrings.TWideStringList を使うと楽。
 - Delphi 2005 から存在する。
- Unicode 版 Delphi では ANSI 形式ファイルは TEncoding で扱う。
- Unicode 版 Delphi に TAnsiStringList は 存在しない。
 - ANSI 版 Delphi の RTL から TStrings / TStringList クラスをコピーして TAnsiStrings / TAnsiStringList へ置換して使う。
 - JCL の JclAnsiStrings.TJclAnsiStringList を使う。
 - もっとも、ファイル入出力のためだけに AnsiString 版の TStringList を使う事はまずないと思われる。

ファイル入出力

```
// TEncoding を使った例 (Unicode 版 Delphi) - UTF-8
var
  SL: TStringList;
  S: String;
begin
  SL := TStringList.Create;
  try
    SL.LoadFromFile('C:¥UTF8.TXT', TEncoding.UTF8);
    S := SL.Text;
  finally
    SL.Free;
  end;
  ShowMessage(S);
end;
```

```
// TEncoding を使った例 (Unicode 版 Delphi) - EUC-JP
var
  SL: TStringList;
  Enc: TEncoding;
  S: String;
begin
  SL := TStringList.Create;
  Enc := TEncoding.GetEncoding(20932); // EUC-JP
  try
    SL.LoadFromFile('C:¥EUCJP.TXT', Enc);
    S := SL.Text;
  finally
    Enc.Free;
    SL.Free;
  end;
  ShowMessage(S);
end;
```


CSV / INI ファイル

- CSV ファイルは ANSI または UTF-8 で出力する
 - Microsoft Excel は UTF-16 形式の CSV ファイルを正しく読み込まない。
- INI ファイル (Unicode 版 Delphi)
 - TIniFile の場合、ファイルが存在しない場合に INI ファイルを出力すると ANSI になってしまう
 - 無難に UTF-16 で INI ファイルを保存したい場合には TMemIniFile を使って必ずエンコーディング指定 (TEncoding.Unicode) を行う
- TStringList
 - 明示的にエンコーディング指定を行う
(TMemIniFile と TStringList はエンコーディングを指定しないと OS デフォルトの ANSI コードページが使われる)

文字列操作(サロゲートペア) - Unicode

- Delphi でサロゲートペアを処理するには？
 - 一旦、UCS4String (UTF-32) へ変換してから文字列操作し、UnicodeString / WideString に戻す。
 - Shift_JIS の 2 バイト文字の検出と同じように ByteType() を使って走査してもいいが、面倒な事になる。
 - UCS4String は動的配列なので、Copy() は使える。
(動的配列は添字が 0 から始まる事に注意)
 - RTL の UnicodeStringToUCS4String() / WideStringToUCS4String() では、変換時にサロゲートペアが正しく処理されないので注意が必要。
 - MECSUtils (<http://cc.embarcadero.com/item/26061>) を使う。
サロゲートペアを考慮した MecsCopy() 等が使える。

文字列操作(サロゲートペア) - Unicode

```
// USC4String を使った例
var
  U4_1, U4_2: UCS4String;
  S: String;
begin
  S := #$20BB7' 野家';

  // UTF16 -> UTF32
  //U4_1 := UnicodeStringToUCS4String(S); // not work
  U4_1 := MECSUtils.UTF16ToUTF32(S);

  // UCS4String は 0 ベースな事に注意
  U4_2 := Copy(U4_1, 1, 1);

  // ヌル終端を作成
  SetLength(U4_2, Length(U4_2) + 1);
  U4_2[Length(U4_2) - 1] := 0;

  // UTF32 -> UTF16
  //S := UCS4StringToUnicodeString(U4_2); // not work
  S := MECSUtils.UTF32ToUTF16(U4_2);

  // '野' が表示される
  ShowMessage(S);
end;
```

```
// MecsUtils を使った例
var
  S: String;
begin
  S := #$20BB7' 野家';

  // 2 文字目をコピー
  S := MecsUtils.MecsCopy(S, 2, 1);

  // '野' が表示される
  ShowMessage(S);
end;
```

文字列操作(正規化) - Unicode

- Delphi で正規化を行うには？
 - RTL には正規化のための関数/クラスは用意されていない。
 - normaliz.dll の NormalizeString() API を使う。
XP / 2003 の場合、
Microsoft Internationalized Domain Names (IDN) Mitigation APIs 1.1
<http://www.microsoft.com/downloads/details.aspx?FamilyID=ad6158d7-ddba-416a-9109-07607425a815&displaylang=en>
または Internet Explorer 7 以降が必要。
Vista / 7 では特に必要となるものはない。
 - 上記に該当しない場合には (Win9x は除く) FoldString() API を使う。
フラグの組み合わせで NFD / NFC / NFKD / NFKC 相当の正規化が可能。
 - 実装が面倒なら、MECSUtils の MecsNormalize() を使う。

文字列操作(正規化) - Unicode

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S, N: WideString;
function ShowBinary(S: WideString): WideString;
var
  i: Integer;
begin
  result := S + ':'#$000D#$000A;
  for i:=1 to Length(S) do
    result := result + Format('U+%.4x ', [Word(S[i])]);
end;
begin
  S := 'ガギグゲゴ';
  // NFD
  if MecsUtils.MecsNormalize(S, N, NormalizationD) then
    ShowMessage(ShowBinary(N));
  // NFC
  if MecsUtils.MecsNormalize(S, N, NormalizationC) then
    ShowMessage(ShowBinary(N));
  // NFKD
  if MecsUtils.MecsNormalize(S, N, NormalizationKD) then
    ShowMessage(ShowBinary(N));
  // NFKC
  if MecsUtils.MecsNormalize(S, N, NormalizationKC) then
    ShowMessage(ShowBinary(N));
end;
```

MecsNormalize() の戻り値が String でないのは正規化に失敗する事があるから。

また、正規化すると元の文字ではなくなる事があるので、元の文字列は保存しておく必要がある。

DB に格納する場合は、正規化していない文字列と正規化した文字列の双方を格納しなくてはならない事がある。
(Where 句で抽出したい場合)

文字列操作(正規表現) - Unicode

- Delphi で 正規表現を使うには？
 - SKRegExp を使う
<http://komish.com/softlib/skregexp.htm>
 - 日本語の取り扱いが楽だったり、サロゲートペアを意識せずに使える。
また、Perl 互換なので正規表現文字列は Perl のものを流用できる。
 - 外部 DLL 不要。
Delphi で書かれ、ソースコードで提供されている (英語文字列リソースもある)
 - Delphi 2005 またはそれ以降で利用可能
 - ライセンスは MPL (Mozilla Public License)

文字列操作(正規表現) - Unicode

```
uses
    ..., SKRegExpW;

// SKRegExp による正規表現マッチング
if ExecRegExp(' [吉'#$20BB7' ]野[家屋]', Edit1.Text, []) then
    ShowMessage(' Match')
else
    ShowMessage(' No match');
```

入力された文字列が
某有名外食チェーン店の
名前に一致するか？
(誤記対応)

```
// SKRegExp によるマッチング文字列取得
var
    RegExp: TSkRegExp;
begin
    result := '';
    RegExp := TSkRegExp.Create;
    try
        RegExp.Expression := ' [吉'#$20BB7' ]野[家屋]';
        if RegExp.Exec(Edit1.Text) then
            result := RegExp.Match[0];
    finally
        RegExp.Free;
    end;
end;
```

入力された文字列に
某(以下略) の名前が含まれる場合
マッチした部分の文字列を取得。

RegExp.ExecNext を使えば、
マッチする文字列が複数含まれる
場合に繰り返して文字列を抽出で
きる。(SKRegExp のヘルプ参照)

文字列操作 – ラウンドトリップ

- ラウンドトリップ

- 入出力時のファイル変換以外でラウンドトリップを発生させるべきではない。
 - ANSI → Unicode → ANSI
 - Unicode → ANSI → Unicode

- ANSI 版 Delphi では以下の関数を利用すると暗黙的に ANSI → Unicode → ANSI のラウンドトリップが行われる。
 - AnsiMidStr()
 - AnsiRightStr()
 - AnsiLeftStr()
 - AnsiReverseString()

※Delphi 2009 Update 3 以降では修正されている。

※MecsUtils.MecsMidStr() / MecsRightStr() / MecsLeftStr() / MecsReverseString()
で代替可能。

文字列操作 – ラウンドトリップ

```
// ANSI 版 Delphi 用
procedure TForm1.Button1Click(Sender: TObject);
var
  A: String;
  S: String;
  i: Integer;
function ThroughFunc(S: WideString): WideString;
begin
  result := S;
end;
begin
  A := '';
  A := A + #$81#$E6; // ::
  A := A + #$87#$9A; // ::
  A := A + #$FA#$5B; // ::

  // ThroughFuncを通してみる
  A := ThroughFunc(A);

  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' #$. 2x', [Ord(A[i])]);
  ShowMessage(Format(' %s : %s', [A, S]));
end;
```

```
// Unicode 版 Delphi 用
procedure TForm1.Button1Click(Sender: TObject);
type
  SJISString = type AnsiString(932);
var
  A: SJISString;
  S: String;
  i: Integer;
function ThroughFunc(S: String): String;
begin
  result := S;
end;
begin
  A := '';
  A := A + #$81#$E6; // ::
  A := A + #$87#$9A; // ::
  A := A + #$FA#$5B; // ::

  // ThroughFuncを通してみる
  A := ThroughFunc(A);

  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' #$. 2x', [Ord(A[i])]);
  ShowMessage(Format(' %s : %s', [A, S]));
end;
```

文字列操作(FAQ) - Unicode

- 半角 n 文字分で表示させたい / 右寄せするのに半角SP を使いたい
 - EAW の属性テーブルをどうにかする必要がある。
 - 面倒なら、MECSUtils.MecslsFullWidth() で文字幅を調べる。
http://homepage1.nifty.com/ht_deko/tech021.html#MecslsFullWidth
 - 但し、Neutral 属性の文字が含まれると破綻する。
- 半角 n 文字幅分で文字数制限をしたい / 入力制限をしたい
 - 入力された文字列の文字幅を調べて、文字数制限を行う。
 - 但し、サロゲートペアや結合文字列、異体字セレクタ等の制御文字があるので、最大幅で入力した場合の文字列長 (データ長) は一定ではない。
- AnsiString に変換してからバイト数調べればいいのか？
 - Shift_JIS に存在しない文字は “? (0x3F)” に置換されるので、全角サイズであろうと 1 バイトでカウントする事になる...それでもいいのなら。

文字列操作 - ANSI

- Unicode 版 Delphi での AnsiString の文字列操作は基本的には行わない。
 - 可能な限り Unicode からの変換で済ます。
 - AnsiStrings ユニットを uses するのは可能な限り避ける。
- 関数に対して定数を使うときには AnsiString(‘あいう’) のようにキャストして使う (Unicode 版 Delphi)
 - 定数をそのまま使うと、意図せず、UnicodeString 版のオーバーロード関数が使われる事がある。
- ANSI 版 Delphi での AnsiString の文字列操作は Ansi ~ と名のつく RTL で行う。
 - Ansi ~ と名前の付かない関数はマルチバイトを考慮しない。
 - AnsiPosEx() がないのは我慢して。

定数はキャスト (AnsiString)

- Pos 関数での例 (Unicode 版 Delphi)

Case1:

```
var
  A: AnsiString;
  Idx: Integer;
begin
  A := 'あいう';
  Idx := Pos('う', A);
  ShowMessage(IntToStr(Idx))
end;
```

Case2:

```
var
  A: AnsiString;
  Idx: Integer;
begin
  A := 'あいう';
  Idx := Pos(AnsiString('う'), A);
  ShowMessage(IntToStr(Idx))
end;
```

Case1 のコードでは
UnicodeString 版の Pos() 関数が呼ばれ、戻り値は **3** となる。

Case2 のコードのように定数を
AnsiString() でキャストすると、
期待通りに AnsiString 版の
Pos() 関数が呼ばれ、戻り値は **5**
となる。

コンパイラが代入互換性のワーニングを出していないか要確認。

文字コード関係の名前空間

- 文字コード絡みの関数を収録している主な名前空間の一覧
 - System / SysUtils 名前空間は当然なので除外。

| 名前空間 | バージョン | 備考 |
|--------------|---------------------|---|
| HTTPApp | Delphi 3 またはそれ以降 | HTTPDecode() / HTTPDecode() 等 |
| StrUtils | Delphi 6 またはそれ以降 | LeftStr() / MidStr() / RightStr() 等 |
| WideStrings | Delphi 2005 またはそれ以降 | TWideStringList 等 |
| WideStrUtils | | UTF-8 文字列操作等 |
| AnsiStrings | Delphi 2009 またはそれ以降 | AnsiString バージョンの文字列操作関数 |
| Character | | TCharacter クラス ConvertFromUtf32() / ConvertToUtf32() 等 |

- TEncoding や TStringBuilder は SysUtils 名前空間に存在する。
(Unicode 版 Delphi)

AnsiStrings 名前空間 (1)

- Unicode 版 Delphi で、ANSI 版アプリを作るハメになった場合に使う
 - 最初から Unicode 版アプリを設計するのなら極力使わない
 - 局所的に AnsiString を操作する場合には当然使わざるを得ない
- UnicodeString 版と AnsiString 版の関数が存在する場合は注意
 - 同じ名前空間に同名のオーバーロード関数が存在する事がある
 - 型が違うだけの関数が (System, SysUtils, StrUtils) / AnsiStrings に散在している (http://homepage1.nifty.com/ht_deko/tech014.html#tech042)。
 - 一部の関数は、まったく同じものが複数の名前空間に存在する
AnsiLastChar () / CharLength () / NextCharIndex()
これらは AnsiStrings 名前空間と SysUtils 名前空間両方に存在する。
- 意図しない UnicodeString 版関数を使わないよう **AnsiStrings.func()** のように、名前空間を明示する。

AnsiStrings 名前空間 (2)

- ExcludeTrailingPathDelimiter() はUnicodeString 版が呼び出される

```
uses
    ...;

function Test(aPath: AnsiString): AnsiString;
begin
    result := ExcludeTrailingPathDelimiter(aPath);
end;
```

- ExcludeTrailingPathDelimiter() はAnsiString 版が呼び出される

```
uses
    ..., AnsiStrings;

function Test(aPath: AnsiString): AnsiString;
begin
    result := ExcludeTrailingPathDelimiter(aPath);
end;
```


AnsiStrings 名前空間 (3)

- このようにキッチリと名前空間を明示すれば、問題は起きにくい。

```
uses
    ..., AnsiStrings;

function Test(aPath: AnsiString): AnsiString;
begin
    result := AnsiStrings.ExcludeTrailingPathDelimiter(aPath);
end;
```

- 本編にもあった、“ANSI の定数は AnsiString() でキャスト”の話と混同しないように注意する事

任意のコードページの AnsiString

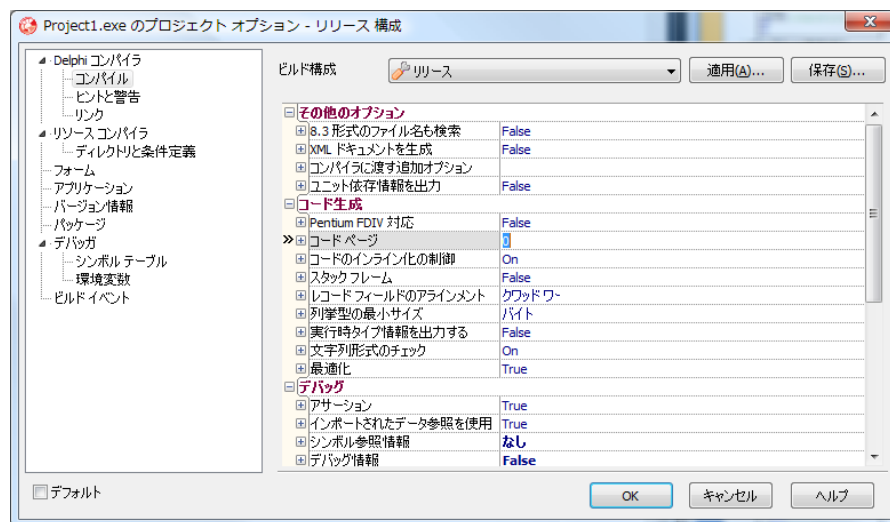
- Unicode 版 Delphi に於いて、以下の定義が可能

```
type
  CP50220 = type AnsiString(50220);
  CP20932 = type AnsiString(20932);
  CP932   = type AnsiString(932);
  CP65000 = type AnsiString(CP_UTF7);
var
  ISO2022JP: CP50220;
  EUCJP: CP20932;
  SJIS: CP932;
  UTF7: CP65000;
  A: AnsiString; // OS デフォルトの ANSI コードページが使われる
begin
  ...
end;
```

- UTF-8 は UTF8String という定義が既に存在する
- 異なるコードページを持つ変数間で代入を行うと、
“暗黙の文字コード(コードページ)変換”が行われる (変換関数は不要)

デフォルトの AnsiString

- OS のデフォルト ANSI コードページに影響される
 - [コントロールパネル | 地域と言語のオプション] “管理”タブの “Unicode 対応ではないプログラムの言語 (Vista の場合)”
- コンパイラオプションにも影響される (0 = OS デフォルト)



- System.DefaultSystemCodePage 変数でも変更できる

コードページ

- 日本語用に使えるコードページ

| コードページ | 名称 | 説明 |
|--------|------------------------------------|--|
| 932 | shift_jis | Windows の SHIFT-JIS。Microsoft 独自の SHIFT-JIS なので、他 OS とは互換性のない文字もある。 |
| 10001 | x-mac-japanese | Mac の SHIFT-JIS。Apple 独自の SHIFT-JIS なので、他 OS とは互換性のない文字もある。 |
| 20290 | IBM290 / x-EBCDIC-JapaneseKatakana | IBM の 日本語 EBCDIC (カタカナ拡張)。 |
| 20932 | EUC-JP | EUC-JP。Microsoft 独自の EUC-JP なので、他 OS とは互換性のない文字もある。 eucJP-ms ではない。 |
| 50220 | iso-2022-jp | ISO-2022-JP と言ったら普通はこの実装。全角文字や半角文字に変化した場合に3バイトのエスケープが発生し、半角カナは存在しない (他コードページの半角カナは全角カナに変換される)。 |
| 50221 | csISO2022JP | 全角文字や半角文字、半角カナに変化した場合に3バイトのエスケープが発生し、半角カナはエスケープシーケンスでエスケープされる。 |
| 50222 | iso-2022-jp | 全角文字や半角文字に変化した場合に3バイトのエスケープが発生し、半角カナは<SI>/<SO>でエスケープされる。 |
| 51932 | EUC-JP | EUC-JP。Microsoft 独自の EUC-JP なので、他 OS とは互換性のない文字もある。 <!> TEncoding には指定できない <!> |
| 65000 | utf-7 | UTF-7 (Unicode) 。CP_UTF7 という定数が用意されている。 |
| 65001 | utf-8 | UTF-8 (Unicode) 。CP_UTF8 という定数が用意されている。 |

RawByteString (1)

- Unicode 版 Delphi で AnsiString 操作関数を作る場合

```
function IsKanjiEscSeq(aStr: AnsiString): Boolean;
begin
    result := (Copy(aStr, 1, 3) = #$1B#$24#$42);
end;

function IsKanjiEscSeq2(aStr: RawByteString): Boolean;
begin
    result := (Copy(aStr, 1, 3) = #$1B#$24#$42);
end;

procedure TForm1.Button1Click(Sender: TObject);
type
    CP50220 = type AnsiString(50220); // ISO-2022-JP
var
    S: CP50220;
begin
    S := 'あいう'; // #$1B#$24#$42 + 'あいう' (+ #$1B#$28#$42)
    if IsKanjiEscSeq(S) then
        ShowMessage('Escape to KANJI')
    else
        ShowMessage('Escape to Other');
end;
```

文字列の先頭を見て ISO-2022-JP のエスケープシーケンスを判断しようとしている。

IsKanjiEscSeq() の引数 aStr は AnsiString なので、ANSI デフォルトコードページである CP932 (Shift_JIS) へ暗黙的に文字コード変換されたものが格納され、意図した通りに動作しない。

IsKanjiEscSeq2() の引数 aStr は RawByteString なので、暗黙的な文字コード変換が行われず、意図した動作となる。

RawByteString (2)

- SetCodePage() の使い方

- 何らかのバイト列を RawByteString へ読み込んだ場合にはコードページが不定なため、SetCodePage() を使ってコードページを設定しなければならない事がある。

```
var
  S: RawByteString;
begin
  S := 'あいう';

  // コンバートを伴わずにコードページを変更
  SetCodePage(S, 932, False);
  ShowMessage(IntToStr(StringCodePage(S)));

  S := 'あいう';

  // コンバートを伴ってコードページを変更
  SetCodePage(S, CP_UTF8); // or SetCodePage(S, CP_UTF8, True);
  ShowMessage(IntToStr(StringCodePage(S)));
end;
```

RawByteString (3)

- 「使うな！」と言っている訳ではない
- Unicode アプリケーションを作る限り、積極的に使わざるを得ないという状況にはまずならない。逆説的に、RawByteString 前提の Unicode アプリケーションは設計を見直した方がいいという事
- 文字列の長さが 0 の場合にはコードページを変更できない
 - 文字列長 0 の場合には、メモリ上に文字列格納用ペイロードが存在しない
- StringCodePage() / StringElementSize() / StringRefCount() / SetCodePage() これら RawByteString に関連する関数の使い方を熟知すべし

コントロール文字列 (1)

- 5 / 6 桁のコントロール文字列
 - Unicodeのコードポイントと同等。U+2000B は “#\$2000B” と記述可能
- 4 桁のコントロール文字列 (“#\$xxxx”)
 - UTF-16 のイメージのまま
 - “#\$D840 + #\$DC0B” のようにしてサロゲートペアを扱う事が可能
- 2 桁のコントロール文字列 (“#\$00..#\$7F”)
 - AnsiChar のコントロール文字列とみなされる
- 2 桁のコントロール文字列 (“#\$80..#\$FF”)
 - コンパイラ指令によって挙動が異なる

コントロール文字列 (2)

- {\$HIGHCHARUNICODE} コンパイラ指令
 - 書式は {\$HIGHCHARUNICODE <ON|OFF>}
 - デフォルトで OFF
- {\$HIGHCHARUNICODE OFF}
 - “#\$80”..“#\$FF” を AnsiChar とみなす
- {\$HIGHCHARUNICODE ON}
 - “#\$80”..“#\$FF” を WideChar とみなす
- コントロール文字列の桁数に注意
 - 桁数を間違えると、Ansi<->Unicode 変換が行われたり、意図しない文字コードを指定する事になってしまうので、桁数は目的を持って設定

コントロール文字列 (3)

- ...なのですが。

```
procedure TForm1.Button1Click(Sender: TObject);
type
  SJISString = type AnsiString(932);
var
  A: SJISString;
  S: String;
  i: Integer;
begin
  // (*1)
  A := #$81#$E6#$87#$9A#$FA#$5B;
  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' 0x%. 2x', [Ord(A[i])]);
  ShowMessage(Format(' %s', [S]));

  // (*2)
  A := #$81#$E6 + #$87#$9A + #$FA#$5B;
  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' 0x%. 2x', [Ord(A[i])]);
  ShowMessage(Format(' %s', [S]));
end;
```

(*1) と (*2) の結果が異なる事に注意。

(*1) の結果は
0x81 0xE6 0x87 0x9A 0xFA 0x5B
となり、正常。

(*2) の結果は
0x81 0xE6 0x81 0xE6 0x81 0xE6
となり、ラウンドトリップが発生している。

“+” での連結の有無によって結果が異なる。

ANSI に対してコントロール文字列を使うのは
避けた方がいい (UTF8String も同様)。

リソースファイル - Unicode

- Unicode 版 Delphi で Unicode 文字列リソースを含んだリソースファイルを作るには CGRC.EXE を使う

```
#define IDS_STRING1      1

STRINGTABLE
BEGIN
    IDS_STRING1  L"あいう"
END
```

```
#define IDS_STRING1      1

STRINGTABLE
BEGIN
    IDS_STRING1  L"¥xD842¥xDFB7野家"
END
```

```
#define IDS_STRING1      1

STRINGTABLE
BEGIN
    IDS_STRING1  L"吉野家"
END
```

- リテラルを L 付きで指定すると Unicode 文字列と解釈される (但し UTF-16)。
- “¥x(16進4桁)”のメタ文字で指定することも可能。
- リソーススクリプトファイル (*.rc) を UTF-8 や UTF-16 で保存すれば、直接サロゲートペアを記述する事が可能。

Unicode の文字列リソース

- リソースコンパイラ / バインダ を使わなくても Unicode の文字列リソースは作れる (Unicode 版 Delphi)。
 - Const の代わりに ResourceString を指定する。
 - 殆どの場合はこれで済むハズ。

```
ResourceString  
str_001 = '吉野家';           // 直接指定  
str_002 = #$D842#$DFB7' 野家'; // コントロール文字列 (UTF-16 表現)  
str_003 = #$20BB7' 野家';     // コントロール文字列 (コードポイント 表現)
```

- ANSI 版 Delphi で ResourceString を用いて文字列リソースを作ると、ANSI 文字列となる。

データベース - Unicode

- Unicode の制限をよく調べる
 - Interbase
UNICODE-FSS では 3 バイトの UTF-8 にしか対応していない。
 - Firebird
2.0 以前の UNICODE-FSS では 3 バイトの UTF-8 にしか対応していない
 - MySQL
現状では 3 バイトの UTF-8 にしか対応していない。
 - MS SQL Server 7.0 / 2000
サロゲートペアを正しく照合しない。
 - Oracle
AL32UTF8 は 4 バイト、**UTF8 は 3 バイト**の UTF-8 。
- DB 格納前に正規化を行う
 - 文字の見た目は同じでも、抽出条件に引っ掛からない場合がある。
 - ただ、なんでもかんでも正規化すればいいというものでもない。

Indy (Unicode 版 Delphi)

- 不具合が出たらとにかく最新版をインストールしてみる
 - 但し、最新版を Delphi 2010 へインストールすると言語の切り替えに対応しなくなる。最新版には日独仏用の文字列リソースファイルが用意されていない。
- TIdHTTP
 - TIdHTTP.Get() は UnicodeString 返すので、そのままの場合によってはバイト列をワード列に単純変換したものが返ってくるため文字化けを起こす。

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  IdHTTP1.Response.CharSet := 'utf-8';  
  Memo1.Lines.Add(IdHTTP1.Get('http://www.embarcadero.com/jp/'));  
end;
```

CharSet を明示的に指定すれば、適切に変換された形で取り込める。

- ただ、この方法だとサイトの文字コードが “Shift_JIS” 等だった場合、ANSI→Unicode 変換が行われるので、問題になる事がある。

Indy (Unicode 版 Delphi)

- TIdHTTP (その2)

- あるがままの状態のもの取得するには、TIdHTTP.Get() のオーバーロードされた手続きを使って TMemoryStream 等に読み込む。

```
procedure TForm1.Button1Click(Sender: TObject);  
var  
    MS: TMemoryStream;  
begin  
    MS := TMemoryStream.Create;  
    try  
        MS.Clear;  
        IdHTTP1.Get('http://www.embarcadero.com/jp/', MS);  
        MS.Position := 0;  
  
        // 処理  
  
    finally  
        MS.Free;  
    end;  
end;
```

これを TMemo 等に LoadFromStream() で読み込んでしまうと、結局 ANSI → Unicode 変換が行われてしまうので注意が必要。

Indy (Unicode 版 Delphi)

- TIdHTTP (その3)

- 下記コードでは Unicode 版 Delphi で EUC-JP のサイトが取得できないのか？

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  IdHTTP1.Response.CharSet := 'EUC-JP';  
  Memo1.Lines.Add(IdHTTP1.Get('http://www.hogehoge.co.jp/'));  
end;
```

- 最新版をちゃんとインストールすれば問題ないはず。
- 面倒臭がりやさんは...
\$(BDS)¥ source¥Indy¥Indy10¥Protocols にある
 - IdGlobalProtocols.pas
 - IdCompilerDefines.incをプロジェクトフォルダにコピーして再構築するといひ。

Indy (Unicode 版 Delphi)

- TIdSMTP
 - ISO-2022-JP / UTF-8 いずれも BASE64 化して送るようにする。
 - QuickSend() は使わない。
 - Subject / Body どちらも BASE64 化する事。
- TIdPOP3
 - BASE64 化されていないメールは場合によってはバイト列をワード列に単純変換したものが返ってくるため文字化けを起こす。この場合、
 - TIdMessage.ContentTransferEncoding()
 - TIdMessage.CharSet()これらを自前で調べて適切な形で取り込むしかない。
- “ANSI 版 Delphi + 古い Indy” で必要だった文字コード変換は不要
 - 文字変換ライブラリを通すと余計に変な事になる。

String をバッファに使うと...

- ANSI バイト列の処理に String をバッファとして使うと？

ANSI 版 Delphi

| | | | | |
|-----------------------------|--------|------|------|------|
| バイト列 | インデックス | 1 | 2 | 3 |
| | 文字 | A | あ | |
| | バイナリ | 0x41 | 0x82 | 0xA0 |
| String バッファ (AnsiString) | インデックス | 1 | 2 | 3 |
| | 文字 | A | あ | |
| | バイナリ | 0x41 | 0x82 | 0xA0 |

バッファを文字列として返しても問題はない

Unicode 版 Delphi

| | | | | |
|--------------------------------|--------|--------|--------|--------|
| バイト列 | インデックス | 1 | 2 | 3 |
| | 文字 | A | あ | |
| | バイナリ | 0x41 | 0x82 | 0xA0 |
| String バッファ (UnicodeString) | インデックス | 1 | 2 | 3 |
| | 文字 | A | [BPH] | [NBSP] |
| | バイナリ | 0x0041 | 0x0082 | 0x00A0 |

バッファを文字列として返すと文字化けする

String をバッファに使うと...

- String をバッファとして使っている場合の対処

| | | | | | | |
|-------------------------|--------|--------|-------|--------|--------|--------|
| バッファ (UnicodeString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | [BPH] | [NBSP] |
| | バイナリ | | | 0x0041 | 0x0082 | 0x00A0 |

↓ 転記 (代入は不可)

| | | | | | | |
|-----------------------|--------|--------|-------|------|------|------|
| 変数 (RawByteString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | ? | ? |
| | バイナリ | | | 0x41 | 0x82 | 0xA0 |

ワード列から
バイト列へシュリンク

↓ コードページを 932 に設定

| | | | | | | |
|-----------------------|--------|--------|-------|------|------|------|
| 変数 (RawByteString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | あ | |
| | バイナリ | | | 0x41 | 0x82 | 0xA0 |

SetCodePage(S, CodePage, **False**) で、バイト列を変換する事なく
任意のコードページに変更する事が可能。

String をバッファに使うと...

- 転記ではなく代入してしまうと？

| | | | | | | |
|-------------------------|--------|--------|-------|--------|--------|--------|
| バッファ (UnicodeString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | [BPH] | [NBSP] |
| | バイナリ | 1200 | 2 | 0x0041 | 0x0082 | 0x00A0 |



代入

| | | | | | | |
|-----------------------|--------|--------|-------|------|------|------|
| 変数 (RawByteString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | ? | ? |
| | バイナリ | 932 | 1 | 0x41 | 0x3F | 0x3F |

ワード列から
バイト列へシュリンク

コードページを 932 に設定

| | | | | | | |
|-----------------------|--------|--------|-------|------|------|------|
| 変数 (RawByteString) | インデックス | コードページ | 要素サイズ | 1 | 2 | 3 |
| | 文字 | | | A | ? | ? |
| | バイナリ | 932 | 1 | 0x41 | 0x3F | 0x3F |

暗黙のコードページ変換により、バイト列も変わってしまう。
この場合、変換するコードページは ANSI デフォルトコードページになる。

String をバッファに使うと...

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  S: String;
  U: UnicodeString;
  A: RawByteString;
begin
  // UnicodeString に CP932 のバイト列を押し込む
  SetLength(U, 3);
  U[1] := #$0041;
  U[2] := #$0082;
  U[3] := #$00A0;

  // 転記
  SetLength(A, Length(U));
  for i:=1 to Length(U) do
    A[i] := AnsiChar(Byte(U[i]));

  // コードページを設定
  SetCodePage(A, 932, False);

  // 確認
  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' 0x%. 2x ', [Byte(A[i])]);
  ShowMessage(S);
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  S: String;
  U: UnicodeString;
  A: RawByteString;
begin
  // UnicodeString に CP932 のバイト列を押し込む
  SetLength(U, 3);
  U[1] := #$0041;
  U[2] := #$0082;
  U[3] := #$00A0;

  // 代入 (NG)
  A := U;

  // コードページを設定
  SetCodePage(A, 932, False);

  // 確認
  S := '';
  for i:=1 to Length(A) do
    S := S + Format(' 0x%. 2x ', [Byte(A[i])]);
  ShowMessage(S);
end;
```

暗黙の文字コード変換を把握する

- 文字欠損の検索

- “暗黙的文字列キャスト(W1057)”
“暗黙的文字列キャストによるデータ喪失の可能性(W1058)”
“set 式で WideChar がバイト文字に変換されました(W1050)”
をワーニングからエラーに昇格させる。

```
{ $WARN IMPLICIT_STRING_CAST ERROR }  
{ $WARN IMPLICIT_STRING_CAST_LOSS ERROR }  
{ $WARN WIDECCHAR_REDUCED ERROR }
```

- AnsiString ⇔ UnicodeString の代入は気付にくい
 - いかな RawByteString でも、UnicodeString からの代入では暗黙の文字コード変換が発生する事に注意。
- Set 式は CharInSet() で置換
 - 置換できない箇所はロジックの見直しが必要という事。

AnsiString のバッファには TBytes

- TBytes = array of Byte なので、暗黙の文字コード変換は行われない
- TBytes はヌル終端で扱う。
 - PAnsiChar / PWideChar へのキャストが楽になる。
- 文字列変数と TBytes の変換
 - Unicode 版 Delphi には TEncoding.GetString() 他 TBytes を扱うクラスや関数が多数存在する。
 - SysUtils.BytesOf() / WideBytesOf() で、文字列変数を TBytes に変換できる。
 - SysUtils.StringOf() / WideStringOf() で、TBytes を文字列変数に変換できる。
 - SysUtils.PlatformBytesOf() / PlatformStringOf () を使う事はまずない。
- なんですけど...

AnsiString のバッファには TBytes

- RTL の注意点

- StringOf() の戻り値は UnicodeString。つまり、AnsiString に対して使うと、暗黙の文字コードが発生してしまう。
- AnsiStringOf() は存在しないので、ANSI バイト列を AnsiString へ変換できない事になり、ラウンドトリップ問題が発生する (QC#83566)。
- BytesOf() / StringOf() は、ANSI バイト列をデフォルト ANSI コードページだとみなして処理する。
- Unicode バイト列は UTF-16 だとみなして処理される。
- 任意の ANSI バイト列が格納されている TBytes に対して BytesOf() / StringOf() を使うことはできない。

- RTL の BytesOf() / StringOf() を使わずに変換するには？

- もちろん、任意の ANSI バイト列を TBytes で扱う方法

AnsiString のバッファには TBytes

```
// AnsiString → TBytes
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  S: String;
  B: TBytes;
  A: AnsiString;
begin
  A := 'ABCあいう';

  SetLength(B, Length(A));
  Move(A[1], B[0], Length(A));

  S := '';
  for i:=0 to Length(B)-1 do
    S := S + Format('0x%.2x ', [B[i]]);
  ShowMessage(S);
end;
```

```
// TBytes → AnsiString
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  S: String;
  A: AnsiString;
  B: TBytes;
begin
  SetLength(B, 10);
  B[0] := $41; // A
  B[1] := $42; // B
  B[2] := $43; // C
  B[3] := $82; // あ
  B[4] := $A0;
  B[5] := $82; // い
  B[6] := $A1;
  B[7] := $82; // う
  B[8] := $A3;
  B[9] := $00; // Null 終端

  A := PAnsiChar(@B[0]);

  S := '';
  for i:=1 to Length(A) do
    S := S + Format('0x%.2x ', [Byte(A[i])]);
  ShowMessage(S);
end;
```


AnsiString のバッファには TBytes

- ANSI を暗黙の文字コード変換せずに ANSI のまま使うには、結局の所、TBytes でバイナリ操作するしかない。
- TBytes は Delphi 2007 またはそれ以降でしか定義されていない。
 - BDS 2006 またはそれ以前ではこう定義しておくといい。

```
{ $IFDEF UNICODE }  
{ $IFDEF VER185 }  
TBytes = array of Byte;  
{ $ENDIF }  
{ $ENDIF }
```

...BytesOf () とか StringOf() とかは使えないけど。

コッパン Vote を要求する！

- QC レポートには文字コードに関連するものがある
 - 修正されないと面倒な事になるものが含まれているので是非 Vote を！
- QC#67468
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=67468>
 - 各種 AnsiString に対して 2 桁のコントロール文字列を用いると、意図した通りに代入されない。
2 桁のコントロール文字列は RawByte 扱いにすべき。
- 回避策
 - `A[1] := #$F0;` のように、1 文字ずつ代入する。
 - ANSI に対してコントロール文字列を使用しない。
 - TBytes や PByte で処理し、Byte のコントロールコードで対処する。

コッパン Vote を要求する！

- QC#67959
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=67959>
 - UnicodeStringToUCS4String() / WideStringToUCS4String() がサロゲートペアを含む文字列を正しく変換しない
- 回避策
 - ConvertToUTF32() / ConvertFromUTF32() で 1 コードポイントずつ変換する。
 - MECSUtils.UTF16ToUTF32() / UTF32ToUTF16() を利用する。
- 備考
 - UTF-32 へ変換してのコードポイント処理が難しくなる。

コッパパン Vote を要求する！

- QC#78134
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=78134>
 - Windows 2000 では WideCharToMultibyte() を利用している RTL のすべてが ISO-2022 系の文字コードを正しく処理しない (TEncoding を含む)。
- 回避策
 - TEncoding.GetEncoding() ではなく、TAltEncoding を利用する。
 - MECSUtils. AnsiToUTF16() / UTF16ToAnsi() を利用する。
 - Indy 10 も同様の問題を抱えている。
“C++Builder 2010 の Indy10 を修正してメールを送信する (山本隆の開発日誌)”
<http://www.gesource.jp/weblog/?p=1786>
を参考に、Indy の修正を行う。
- 備考
 - 根本的な問題は OS のバグ。

コッパン Vote を要求する！

- QC#83508
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=83508>
 - Unicode 版の CharToElementIndex() は、文字列長が 1、インデックスが 1 の場合に正しく動作しない。
- 回避策
 - MECSUtils.MecsCharToElementIndex() を利用する。
- 備考
 - サロゲートペアを考慮した文字列操作が難しくなる。
 - “Delphi 2009 と Unicode : 番外編 (サロゲートペア)”
<http://edn.embarcadero.com/jp/article/38811>
記事中にある、Copy_S() 関数も正しく動作しない。

コッパン Vote を要求する！

- QC#83566
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=83566>
 - AnsiString を返す StringOf() がないため、AnsiString と TBytes を往復変換する事ができない。
- 回避策
 - MECSUtils.MecsStringOf() / MecsAnsiStringOf()
MECSUtils.MecsBytesOf() / MecsAnsiBytesOf() を利用する。
 - 自前で変換する。
- 備考
 - TBytes を AnsiString のバッファとして手軽に利用できない。
 - “Delphi Unicodeワールド パートIII: コードをUnicode対応にする”
<http://edn.embarcadero.com/jp/article/38699>
で推奨されている事を実現するのが難しくなる。

コッパパン Vote を要求する！

- QC#58386
 - <http://qc.embarcadero.com/wc/qcmain.aspx?d=58386>
 - UTF8Encode() はサロゲートペアを正しく処理しない。
 - この QC は Delphi 2009 で Fix されているため、Vote 不可。
- 回避策
 - Unicode 版 Delphi を使う。
 - MECSUtils. UTF16ToUTF8() / UTF8ToUTF16() を利用する。
 - 自前で変換する。
- 備考
 - ANSI 版 Delphi 固有の問題。

MECSUtils

- リファレンスマニュアル
 - http://homepage1.nifty.com/ht_deko/tech021.html
- MECSUtils の目的
 - Unicode 版 Delphi での、サロゲートペアを考慮した文字単位の文字列処理
 - Unicode の正規化 (標準化)
 - ANSI 版 Delphi で、AnsiString に対して Unicode 版文字列処理と同等の事を同名の関数で行う (ソース互換性の向上)。
 - TEncoding が使えない ANSI 版 Delphi での文字コード変換
 - マッピング問題の解決 (日本語コードページのみ)
 - “RTL に存在しない / 使い勝手の悪い” 機能の “補完 / 補助”
- MECSUtils は文字コードに対して万能ではない
 - ご利用は計画的に。

雑多な情報 (Delphi 関連)

- ガリレオ IDE のコードエディタは [右クリック | ファイルフォーマット] でソースファイルの保存形式を変更可能。
- Delphi 2010 でソースファイルの保存形式に [Little Endian UCS-2] を選ぶと、実際には UTF-16LE で保存される (サロゲートペアに対応している)。
- ガリレオ IDE のコードエディタは内部 UTF-8 で処理しているため、保存形式が ANSI だったとしても、ANSI 固有の文字を文字列定数で記述する事はできない。
- ガリレオ IDE のフォームエディタで [右クリック | エディタで表示] すると、日本語等は コードポイント化した Unicode の 10 進数表記で表される。
- *.dfm を日本語のまま編集したい場合には “DFM コンバータエキスパート” が使える。
http://homepage1.nifty.com/ht_deko/junkbox.html#DFMCONVEXP



まとめ

まとめ (Unicode 版 Delphi)

- Unicode アプリケーション作成の基本的な考え方
 - 文字列を表示幅で考えない
 - 固定幅で、見かけ上の 1 文字単位で処理しない
 - ANSI は入出力時のコンバートで対応する
- Web / Mail との連携がある場合
 - 可能な限り UTF-8 で入出力する (内部は UTF-16 なのだからロスレス)
 - ANSI だと厄介な問題が発生する (機種依存 / ラウンドトリップ / マッピング)
- データベースアプリケーションの場合
 - 可能な限り Unicode 系の CharSet を使う
 - ANSI だと厄介な問題が発生する (ラウンドトリップ)

まとめ (ANSI 版 Delphi)

- ANSI アプリケーション作成の基本的な考え方
 - OS のデフォルトコードページと同じ文字コードで処理する
 - Unicode 版 アプリケーションへのマイグレーションを考慮し、“半角 = 1byte” “全角 = 2byte” という前提のコードを排除する。
 - String (AnsiString) をバッファとして使わない
- Web / Mail との連携がある場合
 - Web はともかく、ANSI の Mail は ISO-2022-JP がデフォなので...
 - UTF-8 を使うなら MECSUtils を利用する
- データベースアプリケーションの場合
 - 可能な限り Shift_JIS 系の CharSet を使う

まとめ

- コードページが存在しない ANSI 文字コードはエンコーディングクラスを実装する (Unicode 版 Delphi)。
 - 実装例
 - UCS4Encoding
<http://cc.embarcadero.com/item/26509>
 - CP51932Encoding
http://www.watercolor-city.net/ct_delphi/index.htm
 - IdTextEncoding_ISO2022JP
<http://www.gesource.jp/weblog/?p=1786>
 - エンコーディングクラスを一度実装すれば、TBytes や RawByteString を触る機会は滅多にない。
- 文字コードの問題点と Delphi の問題点は切り分けて考える
 - 何でも Delphi のせいにはしない。
- 時には割り切りも必要。
 - 現実的な“落とし所”を早めに見極める。



Q & A



Extras

資料

- Wikipedia (余裕があれば原典の JIS 規格や RFC もどうぞ)
 - Unicode
<http://ja.wikipedia.org/wiki/Unicode>
 - 東アジアの文字幅
<http://ja.wikipedia.org/wiki/東アジアの文字幅>
 - UTF-16
<http://ja.wikipedia.org/wiki/UTF-16>
 - UTF-8
<http://ja.wikipedia.org/wiki/UTF-8>
 - Shift_JIS
http://ja.wikipedia.org/wiki/Shift_JIS
 - EUC-JP
<http://ja.wikipedia.org/wiki/EUC-JP>
 - ISO-2022-JP
<http://ja.wikipedia.org/wiki/ISO-2022-JP>

資料

- Unihan Database Lookup (Unicode.org)
 - <http://www.unicode.org/charts/unihan.html>
Unicode のコードポイントから文字の詳細情報を検索できます。
- Unicode Character Search (FileFormat.Info)
 - <http://www.fileformat.info/info/unicode/char/search.htm>
Unicode のコードポイントから文字の詳細情報を検索できます。
- 文字コード表 (UIC)
 - <http://charset.uic.jp/>
数多くの文字コード表を閲覧できます。似通った規格の比較も可能です。
- Google 先生
 - “Delphi 2009 Unicode” で検索すると Unicode 版 Delphi の記事がヒットします。

資料

- Code Page Identifiers (Microsoft)
 - <http://msdn.microsoft.com/en-us/library/dd317756%28VS.85%29.aspx>
Windows のコードページの一覧です。
- Go Global Developer Center - Code Pages (Microsoft)
 - <http://msdn.microsoft.com/ja-jp/goglobal/bb964653%28en-us%29.aspx>
Windows の ANSI コードページのコード表です。
- SHIFT-JIS と Unicode 間の変換問題 (Microsoft)
 - <http://support.microsoft.com/default.aspx?scid=kb;ja;170559>
CP932 → Unicode → CP932 ヘラウンドトリップできない文字のリストです。
- Uniscribe (Microsoft)
 - <http://msdn.microsoft.com/ja-jp/library/cc422022.aspx>

資料

- Delphi Unicodeワールド (EDN)
 - <http://edn.embarcadero.com/jp/article/38695>
 - <http://edn.embarcadero.com/jp/article/38698>
 - <http://edn.embarcadero.com/jp/article/38699>
- Delphi 2009 と Unicode (EDN)
 - <http://edn.embarcadero.com/jp/article/38781>
 - <http://edn.embarcadero.com/jp/article/38783>
 - <http://edn.embarcadero.com/jp/article/38786>
 - <http://edn.embarcadero.com/jp/article/38811>
 - <http://edn.embarcadero.com/jp/article/38816>
- Delphi 2009 と 文字列型 (EDN)
 - <http://edn.embarcadero.com/jp/article/38791>

資料

- RawByteString の使い方 (EDN)
 - <http://edn.embarcadero.com/jp/article/39691>
- 文字列とエレメント (EDN)
 - <http://edn.embarcadero.com/jp/article/39692>
- 暗黙の文字列変換 (EDN)
 - <http://edn.embarcadero.com/jp/article/39697>
- DCC 警告W1050の回避方法 (EDN)
 - <http://edn.embarcadero.com/jp/article/39699>
- 文字列型とオーバーロード (EDN)
 - <http://edn.embarcadero.com/jp/article/39706>

資料

- 小形克宏の「文字の海、ビットの舟」 (Internet Watch)
 - <http://www.watch.impress.co.jp/internet/www/column/ogata/index.htm>
- “情報化時代”に追いつけるか？審議が進む「新常用漢字表(仮)」 (Internet Watch)
 - http://internet.watch.impress.co.jp/cda/jouyou_backnumber/2008.html
- 本当は怖い文字コードの話 (gihyo.jp)
 - <http://gihyo.jp/admin/serial/01/charcode>
- 新常用漢字が引き起こす文字コード問題 (ITPro)
 - <http://itpro.nikkeibp.co.jp/article/COLUMN/20091209/341831/>
- Vista文字セット問題(ITPro)
 - <http://itpro.nikkeibp.co.jp/article/COLUMN/20080205/293111/>
- 日本語のUnicodeベンダ依存文字表 (Java Programming Information)
 - <http://www.ingrid.org/java/i18n/unicode.html>

資料

- DEKOのアヤシいお部屋 (DEKO)
 - http://homepage1.nifty.com/ht_deko/
 - Delphi / kylixによるテクニック&アルゴリズム
 - http://homepage1.nifty.com/ht_deko/techalgo.html
 - Delphi の製品情報
 - http://www.geocities.jp/ht_deko/Delphi/index.html
 - MECSUtils
 - <http://cc.embarcadero.com/item/26061>
 - MECSUtils リファレンス
 - http://homepage1.nifty.com/ht_deko/tech021.html
 - DFMコンバータエキスパート
 - http://homepage1.nifty.com/ht_deko/junkbox.html#DFMCONVEXP

資料

- Watercolor City / Infinity (totonica)
 - <http://www.watercolor-city.net/>
 - <http://totonica.s41.xrea.com/>
 - Delphi 2009 特集
 - http://www.watercolor-city.net/ct_delphi/delphi_tiburon/index.htm
 - UCS4Encoding
 - <http://cc.embarcadero.com/item/26509>
 - CP51932Encoding
 - http://www.watercolor-city.net/ct_delphi/index.htm
 - EncodeDetect
 - http://www.watercolor-city.net/ct_delphi/index.htm

資料

- 山本隆の開発日誌 (山本隆)
 - <http://www.gesource.jp/weblog/>
 - UnicodeStringをJIS(ISO-2022-JP)に変換すると文字が減る他
 - <http://www.gesource.jp/weblog/?p=737>
 - <http://www.gesource.jp/weblog/?p=738>
 - <http://www.gesource.jp/weblog/?p=739>
 - <http://www.gesource.jp/weblog/?p=740>
 - <http://www.gesource.jp/weblog/?p=743>
 - <http://www.gesource.jp/weblog/?p=745>
 - <http://www.gesource.jp/weblog/?p=748>
 - <http://www.gesource.jp/weblog/?p=1786>
 - C++ Builder Tips
 - <http://www.gesource.jp/programming/bcb/index.html>

資料

- 小宮秀一の開発日記 (小宮秀一)
 - <http://devdiary.komish.com>
 - SKRegExp
 - <http://komish.com/softlib/skregex.htm>
- Delphi Library (Mr.XRAY)
 - <http://mrxray.on.coocan.jp/index.htm>
 - サンプルプログラム集
 - <http://mrxray.on.coocan.jp/Delphi/plSamples/index.htm>
- Internet Direct (Indy) 関連
 - <http://www.indyproject.org/index.en.aspx>
 - http://homepage1.nifty.com/ht_deko/tech033.html
 - http://homepage1.nifty.com/ht_deko/tech034.html
- Project JEDI (JCL / JVCL)
 - <http://www.delphi-jedi.org/>

資料

- RAD Studio 2010 Migration Center
 - <http://www.embarcadero.com/rad-in-action/migration-upgrade-center>
 - Delphi in a Unicode World (Nick Hodges)
 - <http://www.embarcadero.com/images/dm/technical-papers/delphi-in-a-unicode-world-updated.pdf>
 - Delphi and Unicode (Marco Cantu)
 - <http://www.embarcadero.com/images/dm/technical-papers/delphi-and-unicode-marco-cantu.pdf>
 - Delphi 2009 Unicode (Dr.Bob)
 - <http://www.drbob42.com/examines/examinA7.htm>
 - Delphi Unicode Migration for Mere Mortals (Cary Jensen)
 - <http://forms.embarcadero.com/forms/AMUSCA1003UnicodeJensenWhitepaper>