Unicode in VOTable

IVOA Interop Meeting
Banff
12 October 2014

Mark Taylor

with input from Walter Landry,
Markus Demleitner,
Norman Gray,
Dave Morris,
Pat Dowler,
et al.

$Id: vot-unicode.tex,v 1.6 2014/10/12 15:32:28 mbt Exp $

Outline

- Unicode
- VOTable
- Problem
- Solutions?

Unicode Primer

(Disclaimer: I'm not an expert)

Unicode:

- Represents characters from many character sets
- Each character is a "code point" — an integer
- Code points have designations like U+0058 ("X")
- Code points are arranged in 17 *planes*, each with 65536 code points
  - ▷ BMP (Basic Multilingual Plane), including all normal letters/characters (Latin, Arabic, Chinese, Japanese, Korean, ...)
  - ▷ ... and 16 others for weird characters
  - ▷ No more will be allocated
- There are 1,112,064 valid code points ($= 17 \times 2^{16} - 2048$)
- Most modern languages support Unicode (Python, Java, ...)
- Most old languages do not (FORTRAN 77, C, ...)
- Different "encodings" are defined to serialise a sequence of code points as a byte stream

Unicode Encodings

- UCS-2: exactly 2 bytes for all supported code points
  - Fixed-length code points — easy to handle!
  - Only BMP code points can be represented
  - Obsolete since 1996 (Unicode 2.0)
  - Not supported in modern Unicode-friendly environments?
- UCS-4: exactly 4 bytes for all code points
  - Fixed-length code points — easy to handle!
  - 4 bytes per character — not very efficient
  - Not widely used
- UTF-8: 1–4 bytes for all code points
  - Variable-length code points
  - 7-bit ASCII is identical to UTF-8!
  - Non-ASCII code points require *more than one byte*
  - No endianness issues
- UTF-16:
  - Variable-length code points!
  - Not very efficient for mostly-ASCII text!
  - Endianness!
- ... and many, many more

Unicode Encodings

- UCS-2: exactly 2 bytes for all supported code points
  - ▷ Fixed-length code points — easy to handle!
  - ▷ Only BMP code points can be represented
  - ▷ Obsolete since 1996 (Unicode 2.0)
  - ▷ Not supported in modern Unicode-friendly environments?
- UCS-4: exactly 4 bytes for all code points
  - ▷ Fixed-length code points — easy to handle!
  - ▷ 4 bytes per character — not very efficient
  - ▷ Not widely used
- UTF-8: 1–4 bytes for all code points
  - ▷ Variable-length code points
  - ▷ 7-bit ASCII is identical to UTF-8!
  - ▷ Non-ASCII code points require *more than one byte*
  - ▷ No endianness issues

⇐ This is the useful one

- UTF-16:
  - ▷ Variable-length code points!
  - ▷ Not very efficient for mostly-ASCII text!
  - ▷ Endianness!
- ... and many, many more

VOTable Primer

VOTable 1.3 (and earlier):

- "Strings" in VOTable are arrays of characters
- Characters are one of two `datatypes`:
  - `char`: strict (7-bit) ASCII
  - `unicodeChar`: "UCS-2"
- There are (approximately) 2 data serializations:
  - `TABLEDATA` (XML `<TR>` and `<TD>` elements) — for unicode, encoding as document
  - `BINARY` (stream of bytes) — for unicode, must define encoding
- Arrays (hence strings) may be fixed or variable length.
  - Fixed: `<FIELD datatype="char" arraysize="4">` (length from `arraysize`)

    TABLEDATA: `<TD>IVOA</TD>`

    BINARY:

49	56	4F	41
I	V	O	A

  - Variable: `<FIELD datatype="char" arraysize="*"/>`

    TABLEDATA: `<TD>IVOA</TD>`

    BINARY:

00	00	00	04	49	56	4F	41
				I	V	O	A

    (length from run-length)
  - Length (`arraysize`) is number of characters

VOTable Text

"VOTables support two kinds of characters: ASCII 1-byte characters and Unicode (UCS-2) 2-byte characters. Unicode is a way to represent characters that is an alternative to ASCII. It uses two bytes per character instead of one, it is strongly supported by XML tools, and it can handle a large variety of international alphabets. Therefore VOTable supports not only ASCII strings (datatype="char"), but also Unicode (datatype="unicodeChar")."

VOTable in Practice

You're supposed to put only 7-bit ASCII in char fields

- ... but people sometimes put unicode in there
  (document encoding for TABLEDATA, UTF-8 for BINARY)
- ... and software often copes with it
- ... even though it shouldn't

Problem

VOTable doesn't have proper Unicode support

... so let's declare char to be Unicode!

- In legal VOTables, only 7-bit ASCII characters are present in char
- TABLEDATA serialization: looks after itself (using XML unicode machinery)
- BINARY serialization: 7-bit ASCII characters are the same in UTF-8 and ASCII, so for characters that are legal now, existing en/decoding methods will work as before

  ☺ No problem?

With UTF-8, number of characters doesn't tell you byte count

49	56	4F	41
I	V	O	A

49	56	CE	A9	41
I	V	Ω		A

☹ Problem.

- ▷ In the BINARY serialization, array length no longer tells you how many bytes are in the field
- ▷ If you have the number of code points, you have to read the bytes in a BINARY byte stream to work out how many bytes are present.
- ▷ This means you can't skip parts of stream (do pointer arithmetic) → inefficient
  (char fields are not known length, rows are not fixed length even if all fields are)

Possible Solutions

- No change
  - No unicode allowed in char (but people will keep putting it there)
  - UCS-2 in unicodeChar (but it's obsolete, not supported by software, can't represent wacky characters, and is not widely used)
- Use UCS-4 for BINARY
  - Inefficient (4 bytes per character)
  - (also eccentric and endianness to cope with)
- Use UTF-8 for BINARY
  - Some difficulties relating to field length

UTF-8 Questions

Use UTF-8 for BINARY encoding

- What datatype?
  - `datatype="char"`: matches common current (illegal) usage, it's the most obvious type to use for "normal" text, but some backward compatibility issues
  - `datatype="utf-8"`?: sounds somehow non-standard; if so, do we deprecate/remove char?
- What about `unicodeChar` datatype?
  - Deprecate? Remove?
- How to handle array length?

  P1: Define both arraysize and binary run-length as *number of code points*
  $\longrightarrow$ can't do pointer arithmetic on BINARY streams

  P2: Define arraysize as *number of code points* and binary run-length as *number of bytes*
  $\longrightarrow$ need run-length even for fixed-length char arrays; can do some pointer arithmetic (to skip a unicode field you need to read the run-length, but not the characters)

  P3: Define both arraysize and binary run-length as *number of bytes the characters would take in UTF-8*
  $\longrightarrow$ declared arraysize N does not guarantee you can store N code points