Charset, i18n PHP, Javascript e MySQL

i18n e l10n nel software

- IBM e Sun: globalization
- Microsoft: World-Readness
- NLS (Native language support)
- Linguaggio UI, input, display, time, currency, numbers
- Localized, Multilingual, Internationalized, Full internationalized

Aspetti della i18n e l10n

- Lingua
  - Testo codificato
    - alfabeti / scripts (Unicode)
    - simboli numerali
    - direzione di scrittura
    - text layout complessi
    - processazione del testo (capitalization, collations)
    - forme plurali
  - Input
    - Keyboard shortcut e keyboard layouts
  - Rappresentazione grafica del testo
  - Audio
  - Sottotitolazione dei film e dei video
- Cultura
  - Immagini e colori
  - Nomi e titoli
  - Numerazioni assegnate dai governi
  - Numeri di telefono e codici postali
  - Valuta
  - Pesi e misure
  - Misure della carta
- Convenzioni di scrittura
  - Formattazione di date e ore
  - Time zones
  - Formattazione dei numeri (decimal separator, digit grouping)
  - Differenze nei simboli (quoting)
- Ogni altro aspetto del software soggetto a conformità normative
  - Leggi e regolamenti del paese (es. confini su dispute territoriali)

Locale

- Insieme di parametri che definiscono la lingua, il paese e una serie di varianti di localizzazione che hanno effetto su alcune chiamate e librerie di sistema
- POSIX (Unix, Linux); BCP 47
  [language[_territory][.codeset][@modifier]]
  C, en_US, en_US.utf8, it_IT.iso885915@euro
- Impostazioni locale
  - formattazione dei numeri
  - classificazione dei caratteri, case conversion
  - date-time formatting
  - string collation
  - currency format
  - paper size
  - altro...

POSIX (ISO 15897)

```
LANG=it_IT.UTF-8
LC_CTYPE="it_IT.UTF-8"
LC_NUMERIC="it_IT.UTF-8"
LC_TIME="it_IT.UTF-8"
LC_COLLATE="it_IT.UTF-8"
LC_MONETARY="it_IT.UTF-8"
LC_MESSAGES="it_IT.UTF-8"
LC_PAPER="it_IT.UTF-8"
LC_NAME="it_IT.UTF-8"
LC_ADDRESS="it_IT.UTF-8"
LC_TELEPHONE="it_IT.UTF-8"
LC_MEASUREMENT="it_IT.UTF-8"
LC_IDENTIFICATION="it_IT.UTF-8"
```

- Microsoft (LCID) è un numero (p.e. 1040 decimale: Italia)
- Windows Vista, nuove funzioni che utilizzano BCP 47

CLDR

- Common Locale Data Repository (v. 22, 10/09/2012)
- Progetto dell'Unicode Consortium, supera molte delle limitazioni dei locale POSIX, è una panacea per l'i18n del software
- DB in formato XML (LDML, UTS #35, BCP 47 + RFC 6067 + l.subtag)
- Usato in Mac OS X, OpenOffice, IBM AIX, Google, Android, e altri..
- Un POSIX locale può essere dedotto da CLDR (sottoinsieme)
- Tipi di dati:
  - Traduzioni dei nomi delle lingue e degli script
  - Traduzioni di territori e paesi
  - Traduzioni di nomi di valuta, dei sistemi di numerazione, dei calendari
  - Traduzioni di giorni della settimana, mesi, ere, periodi del giorno, in forma piena e abbreviata
  - Traduzioni e pattern di formattazione delle timezone e città di esempio
  - Traduzioni delle parti del calendario (mese, trimestre, etc..)
  - Pattern per la formattazione di data e ora, tempi relativi
  - Repertorio dei caratteri usati per scrivere nella lingua
  - Pattern per la formattazione dei numeri e delle valute
  - Regole di collazione
  - Regole di formattazione dei numeri in sistemi tradizionali (numeri romani, numeri armeni)
  - Regole per la compitazione dei numeri
  - Regole di traslitterazione tra script
  - Regole per gli elenchi, per il quoting, per la suddivisione del testo

ICU

- International Components per Unicode (v. 50.0.2, 10/09/2012)
- Primi anni ’90, nasce da un progetto Taligent, poi IBM e Sun
- Librerie C/C++ e Java per supporto di Unicode e i18n software
- Funzioni fornite:
  - Gestione del testo Unicode, proprietà complete dei caratteri, conversione di charset
  - Regexp Unicode, limiti di carattere, parola, linea
  - Ricerca e collazione lingua dipendente
  - Localizzazione tramite CLDR
  - Layout e grafemi complessi (arabo, ebraico, script indiani e thai)
  - Calendari e time zones
  - Formattazione e parsing di date, orari, numeri, valute, messaggi

gettext

- GNU gettext è un framework di localizzazione: è lo standard *de facto* della localizzazione in ambito open source / free software
- Utilizza traduzioni a livello di messaggio: ogni messaggio è tradotto individualmente
- PHP: **`__('message'), _e('message')`**
- Tipi di File:
  - .POT (template file)
  - .PO (formato testo per una specifica lingua)
  - .MO (formato bianario utilizzato dalle librerie)
- Poedit, GNU Gettext

iconv

- Programma Unix / Linux per la conversione tra charset
- Conversione da stdin:

  `iconv -f iso-8859-1`
- Conversione di file
  `iconv -f iso-8859-1 -t utf-8 <infile >outfile`
- Lista dei charset supportati
  `iconv –l`
- Disponibile come libreria di molti altri linguaggi di programmazione (PHP)
- Flag **`//TRANSLIT`** e **`//IGNORE`**

Unicode character escaping

- HTML hexadecimal numeric character references (e.g. é)
- HTML decimal numeric character references (e.g. é)
- HTML character entities (e.g. é)
- SGML hexadecimal numeric character references (e.g. \#x00E9;)
- SGML decimal numeric character references (e.g. \#0233;)
- \u-escaped hexadecimal, as used in Python and Java (e.g. \u00E9)
- \u-escaped hexadecimal within the BMP, \U-escapes beyond the BMP, (e.g. \u00E9 but \U00010024) as used in Tcl and Scheme.
- \u-escaped decimal (e.g. \u0233) as used in Rich Text Format
- U+-escaped hexadecimal (e.g. U+00E9) as in the Unicode standard
- U-escaped hexadecimal (e.g. U00E9)
- u-escaped hexadecimal (e.g. u00E9)
- U-escaped hexadecimal within angle brackets (e.g. <U00E9>) as used in POSIX locale specifications
- \x-escaped hexadecimal (e.g. \x00E9) as used in Tcl for numbers as opposed to characters
- \x-escaped hexadecimal with braces (e.g. \x{00E9}) as used in Perl
- hexadecimal within single quotes with prefix X (e.g. X'00E9')
- RFC 2396 URI format (e.g. %C3%A9)
- RFC 2045 Quoted Printable (=-escaped hexadecimal UTF-8) e.g. =C3=A9
- \-escaped octal UTF-8 (e.g. \303\251)
- Hexadecimal UTF-8 with each byte enclosed in angle brackets (e.g. <C3><A9>)
- Standard hexadecimal (e.g. 0x00E9)
- Raw hexadecimal (e.g. 00E9)
- Common Lisp hexadecimal format (e.g. #x00E9)
- Perl v-prefixed decimal format (e.g. v233)
- Hexadecimal numbers preceded by "$" (e.g. $00E9).
- Hexadecimal numbers preceded by "16#" (e.g. 16#00E9) as in Postscript.
- Hexadecimal numbers preceded by "#16r" (e.g. #16r00E9) as in Common Lisp.
- Hexadecimal numbers preceded by "16#" and followed by "#" (e.g. 16#00E9#) as in ADA.
- OOXML hexadecimal numbers preceded by "_x" and followed by "_". (e.g. _x00E9_)
- Hexadecimal numbers preceded by "%u" (e.g. %u00E9)

PHP

Il codice PHP ha un charset?

- La domanda è mal posta, tu volevi dire: *maestro, che ore sono?*
- Un file sorgente PHP è una successione di byte
- Tutte le parole chiave del linguaggio e i simboli speciali (operatori, delimitatori, etc.) sono caratteri ASCII (range 00-7F)
- Per gli identificatori e le stringhe si possono usare anche gli altri byte (con determinate regole per gli identificatori)
- Encoding che si possono usare per i sorgenti PHP:
  - Un single byte encoding ASCII-compatibile (range 00-7F)
  - Un multiple byte encoding che
    - abbia single byte ASCII-compatibili nel range 00-7F
    - non utilizzi sequenze di escape ISO 2022
    - non utilizzi byte nel range 00-7F in nessuna delle sequenze multibyte utilizzate dall'encoding
- Esempi di encoding usabili:
  - ASCII, ISO-8859-1, Windows CP 1252, UTF-8
- Encoding non usabili:
  - UTF-16, JIS, SJIS, ISO-2022-JP, BIG5
- L'interprete PHP non ha cognizione dell'encoding usato dal file sorgente, pertanto l'interpretazione dei byte nelle stringhe come caratteri dipende da come viene interpretato l'output

Coerenza di charset

- Un file PHP salvato in ISO-8859-1 contiene il seguente codice:

```
<?
setlocale(LC_ALL,
"zh_CN.gb2312");
header("Content-type: text/html;
charset=UTF-8");
mb_internal_encoding("ISO-8859-
9");
...
mysql_query("SET NAMES 'ISO-8859-
2'");
```

PHP e BOM UTF-8

- Un file PHP normalmente è intervallato da codice di programmazione e output (anche se questo non è un bel modo di scrivere il codice, ma questa è un'altra storia)
- Il protocollo HTTP prevede una serie di intestazioni e un corpo, sia nelle richieste che nelle risposte. Le intestazioni devono venire prima del corpo.
- L'output PHP fa parte del corpo della risposta
- Le funzioni PHP che producono intestazioni HTTP (header, setcookie) devono quindi venire **prima** di qualsiasi output, a meno che non utilizziate le funzioni ob (output buffering)
- Il BOM UTF-8 è una sequenza di 3 byte all'inizio del file sorgente, prima di **qualunque** tag di apertura del codice PHP (quindi anche prima di qualunque dichiarazione ob_)
- L'interprete PHP non sa niente del BOM e lo considera output lecito: le intestazioni non possono più essere mandate
- Quindi: se salvate i vostri file PHP in UTF-8 fatelo **senza il BOM**

Funzioni single byte sulle stringhe

- Tutte le funzioni di manipolazione delle stringhe “classiche” PHP sono funzioni single byte:
  1 carattere = 1 byte
- strlen, strpos, substr, etc..
- Falliscono (danno risultati imprevisti) con encoding multibyte come UTF-8
- Per esempio (ipotizzo sorgente in UTF-8):
  - **`strlen("è")`**: 2
  - **`substr("Anna è bella",0,6)`**: Anna **`&#xFFFD;`**
- Il comportamento di alcune è influenzato dalle impostazioni di localizzazione
  - **`setlocale(LC_ALL,"it_IT.ISO88591");`**
    `strtolower("È")`: **`&#xFFFD;&#xFFFD;`** (E3 88: ãˆ)
  - **`setlocale(LC_ALL,"it_IT.UTF8");`**
    `strtolower("È")`: È (C3 88: Ãˆ)

htmlentities & co.

- **ACHTUNG**!! L‘encoding di default di queste funzioni è cambiato in PHP 5.4
- **`htmlentities`** (4 argomenti)
  - stringa input
  - flags (ENT_COMPAT, ENT_QUOTES, ...)
  - encoding (default UTF-8 da 5.4.0)
  - double encode
- **`htmlspecialchars`** (come la precedente ma limitata alle 5 entities XML [ampersand , double quote, single quote, less than, greater than]
- **`html_entity_decode`**, **`htmlspecialchars_decode`** (funzioni inverse)
- Le entities sono incomplete... per avere la lista esatta supportata dalla vostra versione di PHP:
- **`get_html_translation_table`**

utf8_encode, utf8_decode

- Anche se il nome è accattivante, sono molto limitate
- Effettuano solamente la conversione da ISO-8859-1 a UTF-8... ...e viceversa
- Se il vostro input è, per esempio, Windows-1252 NON aspettatevi buoni risultati. Ad es.:
- **`utf8_encode("\x93smart quotes\x94")`**
- Le smart quotes “ ” (93 e 94 esadecimale in Windows-1252) verranno convertite nei C1 controls ISO (invisibili in UTF-8)
- **`iconv("cp1252","utf-8","\x93smart quotes\x94")`**

setlocale

- **`setlocale`**: imposta le informazioni di localizzazione (env vars) **`LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, LC_MESSAGES`**
- Viene alterato il comportamento di **alcune** librerie C sottostanti (per esempio strotlower, strtoupper, strftime, gettext, etc..). NON IMPOSTA LA TIMEZONE!
  - **`setlocale(LC_CTYPE,"tr_TR.ISO88599");`**
  - **`echo strtoupper("i");`** (produce İ: \xDD)
  - **`setlocale(LC_TIME,"it_IT.ISO88591");`**
  - **`echo strftime("%B",0);`** (potrebbe produrre Dicembre o Gennaio [!!Timezone])
- Problematico in ambiente Windows
- **`nl_langinfo`**: restituisce informazioni specifiche della localizzazione corrente: **`nl_langinfo(ABDAY_3)`**
- **`localeconv`**: restituisce (quasi) TUTTE le informazioni della localizzazione corrente in un array associativo

Supporto al linguaggio umano e codifiche di carattere

- Esistono in PHP una serie di estensioni specifiche:
  - Enchant (spelling)
  - FriBiDi (bidirectonal)
  - Gender
  - Gettext
  - iconv
  - intl
  - Multibyte String
  - Pspell
  - Recode

iconv in PHP

- **`iconv_get_encoding`** — Retrieve internal configuration variables of iconv extension
- **`iconv_mime_decode_headers`** — Decodes multiple MIME header fields at once
- **`iconv_mime_decode`** — Decodes a MIME header field
- **`iconv_mime_encode`** — Composes a MIME header field
- **`iconv_set_encoding`** — Set current setting for character encoding conversion
- **`iconv_strlen`** — Returns the character count of string
- **`iconv_strpos`** — Finds position of first occurrence of a needle within a haystack
- **`iconv_strrpos`** — Finds the last occurrence of a needle within a haystack
- **`iconv_substr`** — Cut out part of a string
- **`iconv`** — Convert string to requested character encoding
- **`ob_iconv_handler`** — Convert character encoding as output buffer handler

Le funzioni multibyte

- Forniscono un rimpiazzo alle funzioni single byte
- In generale hanno lo stesso nome delle funzioni single byte con “**`mb_`**” preposto al nome della funzione (es.: **`mb_strlen`**)
- Supportano molti encoding (meno di iconv)
- Le impostazioni **non** dipendono dal locale, ma dall’internal encoding delle mb (e dal “language”)
- Attenzione: l’internal encoding di default delle mb è Latin1. Può essere modificato a livello server, ma è sconsigliabile su hosting condiviso
- Forniscono anche funzioni specifiche, cioè senza un corrispettivo single byte: per esempio per la conversione fra charset, per la codifica delle intestazioni MIME come Encoded words, e altro ancora

mb runtime configuration

- mbstring.language
- mbstring.detect_order
- mbstring.http_input
- mbstring.http_output
- **mbstring.internal_encoding**
- mbstring.script_encoding
- mbstring.substitute_character
- **mbstring.func_overload**
- mbstring.encoding_translation
- mbstring.strict_detection

Function overloading

mbstring.func_overload	original function	overloaded function
1	mail()	mb_send_mail()
2	strlen()	mb_strlen()
2	strpos()	mb_strpos()
2	strrpos()	mb_strrpos()
2	substr()	mb_substr()
2	strtolower()	mb_strtolower()
2	strtoupper()	mb_strtoupper()
2	stripos()	mb_stripos()
2	strripos()	mb_strripos()
2	strstr()	mb_strstr()
2	stristr()	mb_stristr()
2	strrchr()	mb_strrchr()
2	substr_count()	mb_substr_count()
4	ereg()	mb_ereg()
4	eregi()	mb_eregi()
4	ereg_replace()	mb_ereg_replace()
4	eregi_replace()	mb_eregi_replace()
4	split()	mb_split()

Funzioni mb specifiche

- Determinazione di charset (inaffidabile)
  - **`mb_detect_encoding('perch\xE9', 'UTF-8, ISO-8859-1');`** (UTF-8 !?!)
- Validazione di charset (problemi)
  - per validare utf-8 meglio usare questa: **`$valid_utf8 = (@iconv('UTF-8','UTF-8',$string) === $string);`**
- Conversione di charset
  - iconv supporta più encoding
- Codifica di intestazioni MIME
- Regular expressions (inaffidabile)

intl

- Wrapper ad ICU in PHP per il supporto delle funzioni fornite da queste librerie
- Bundled con PHP 5.3.0
- Installabile come modulo a parte in PHP > 5.2.0 (5.2.4+ raccomandato)
- Collator
- NumberFormatter
- Locale
- Normalizer
- MessageFormatter
- IntlDateFormatter
- ResourceBundle
- Spoofchecker
- Transliterator
- Grapheme Functions
- IDN Functions
- intl Functions

PCRE e Unicode

- PCRE: Perl compatible regular expressions
- Escape sequences:
  - Unicode properties:
    - \p{xx} (un carattere con la proprietà Unicode xx)
    - \P{xx} (un carattere senza la proprietà Unicode xx)
    - http://www.php.net/manual/en/regexp.reference.unicode.php
    - In determinate condizioni si può usare il nome di uno script in {xx}, ad esempio \p{Han}
  - Code points:
    - \xhh hh: 2 cifre esadecimali del carattere o del byte della stringa
    - \x{...} stringa di cifre esadecimali del code point
- PCRE UTF-8 mode pattern modifier: **u**
  (incompatibile con Perl, tratta la stringa di input come UTF-8)
- Senza il pattern modifier **u** l'escape sequence \xhh viene considerata in byte
- \C : match di un singolo byte in UTF-8 mode (. matcha un **carattere**)
- Esempio (sorgente e intestazioni HTTP UTF-8):

```
$string = "\xC3\xA8 perch\xC3\xA9 è così";
echo $string."<br/>";
preg_match_all('/[\x{E8}-\x{E9}]+/u', $string, $matches);
print_r($matches);
```

è perché è così
Array ([0] => Array ([0] => è [1] => é [2] => è))

Accesso ai file, PHP e charset

- Unix, Linux: i nomi dei file sono stringhe di byte terminata dal byte 00, l‘encoding dipende dalla variabile di ambiente LC_CTYPE
- I nomi dei file possono pertanto essere in qualunque codifica: dipende dall‘utente (dal processo) che crea i file
- Quindi per accedere ad un file basta conoscere il suo nome come stringa di byte, indipendentemente dall‘encoding, oppure più banalmente sapere come è stato creato
- Windows utilizza UTF-16 (a partire da Win2000, NT utilizzava UCS-2, DOS e Win95 la Windows code page)
- Tuttavia l‘interprete PHP in Windows utilizza le Windows Code Page (non Unicode programs nelle Regional settings)
- Per accedere a un file, PHP deve convertire la stringa dall‘encoding interno di rappresentazione al Windows code page impostato nelle Regional settings del sistema
- Alcuni file potrebbero non essere mai raggiungibili
- http://www.icosaedro.it/articoli/php-file-name.html

Nome del file nel file upload

- In una form con enctype multipart/form-data l'utente puà inviare un nome di file che contiene caratteri non ASCII
- Il browser invia il nome del file secondo i meccanismi (e gli eventuali bachi) già visti come gli altri campi della form
- Il sistema operativo del client e del server possono essere diversi, e utilizzare caratteri riservati diversi nei nomi dei file
- Ci possono essere limitazioni diverse nella lunghezza dei nomi dei file e altre caratteristiche (case sensitivity)
- In ogni caso / e \ non si possono utilizzare nei nomi dei file caricati via file upload PHP (il filename viene troncato nell'array $_FILES)
- Occorre sanitizzare l'input utente
- Non conviene salvare il file sul filesystem con il nome originale uploadato dall'utente
- http://www.icosaedro.it/articoli/php-file-upload.html

Nome del file nel file download

- Da PHP è possibile suggerire il nome di un file da scaricare con l'intestazione HTTP Content-Disposition
  `header("Content-Disposition: attachment; filename=\"$file_name\"");`
- Le intestazioni HTTP dovrebbero contenere solo caratteri ASCII, in ogni caso manca l'informazione sull'encoding
- RFC 2231: meccanismo analogo all'Encoded word di MIME
  `header("Content-Disposition: attachment; filename*=UTF-8'" . rawurlencode($file_name));`
- I browser manifestano comportamenti completamente diversi
- Questa soluzione funziona con tutti i moderni browser tranne Internet Explorer (versioni vecchie di Opera comunque manifestano diversi problemi)
- Con IE funziona il raw urlencoding del file name UTF-8, che però non va bene con Firefox (Chrome OK entrambi)
- Occorre differenziare lato server
- Stesse considerazioni dell'upload: sysop client e server possono essere diversi, utilizzare caratteri riservati diversi nei nomi dei file, avere limitazioni diverse nella lunghezza dei nomi dei file e altre caratteristiche
- http://www.icosaedro.it/articoli/php-file-download.html

Javascript

- Come il PHP i sorgenti javascript sono fatti di byte
- Un file js esterno non prevede dichiarazioni di charset in-document
- In mancanza di dichiarazioni lato server il browser interpreta le stringhe (e gli identificatori) del javascript nella codifica del documento referente
- Internamente le stringhe del Javascript sono in UTF-16 (usa i surrogati)
- Tuttavia la lunghezza delle stringhe è in termini di code points
- I caratteri fuori dal BMP sono lunghi 2 caratteri
- Di fatto è un UCS-2 che consente i surrogati

escape, unescape, encodeURI(component)

- La funzione **`escape`** di Javascript implementa un “cattivo” Urlencoding
- Alcuni caratteri non url safe (ma anche ~) vengono percent encoded (~!#$%^&(){}[]=:,;?'"\)
- Caratteri inclusi in Latin-1 non ASCII vengono convertiti in Latin-1 e percent encoded (è = %E8)
- Caratteri in BMP fuori da Latin-1 vengono escaped con la sequenza %uHHHH (€ = %u20AC)
- Caratteri fuori da BMP vengono escaped come coppia di code point surrogati
- La funzione **`encodeURI`** effettua il percent encoding di %^{}[]"\
- **`encodeURIComponent`** effettua il percent encoding di @#$%^&{}[]=:/,;?+"\
- Per caratteri non ASCII entrambe effettuano la conversione a UTF-8 e il percent encoding

AJAX

- Tutti i browser moderni (IE7+, Firefox, Chrome, Safari e Opera) supportano il built-in XMLHttpRequest
- TUTTE le richieste effettuate da questo oggetto inviano i dati in UTF-8
- A prescindere dalla codifica della pagina chiamante
- A prescindere dalle proprietà che è possibile impostare per l’oggetto
- Se lo script che processa la richiesta AJAX si aspetta di lavorare con un altro encoding, deve applicare una transcodifica

Stringhe in JSON: RFC 4627

The representation of strings is similar to conventions used in the C family of programming languages. **A string begins and ends with quotation marks**. All Unicode characters may be placed within the quotation marks except for the characters that **must be escaped**: **quotation mark, reverse solidus, and the control characters** (U+0000 through U+001F).

Any character may be escaped. If the character is in the Basic Multilingual Plane (U+0000 through U+FFFF), then it may be represented as a **six-character sequence**: a reverse solidus, followed by the lowercase letter u, followed by four hexadecimal digits that encode the character's code point. The hexadecimal letters A though F can be upper or lowercase. So, for example, a string containing only a single reverse solidus character may be represented as "\u005C".

Alternatively, there are **two-character sequence escape representations of some popular characters**. So, for example, a string containing only a single reverse solidus character may be represented more compactly as "\\".

To escape an extended character that is not in the Basic Multilingual Plane, the character is represented as **a twelve-character sequence, encoding the UTF-16 surrogate pair**. So, for example, a string containing only the G clef character (U+1D11E) may be represented as "\uD834\uDD1E".

Escaping in JSON: RFC 4627

```
string = quotation-mark *char quotation-mark

char = unescaped /
       escape (
           %x22 /          ; "    quotation mark  U+0022
           %x5C /          ; \    reverse solidus U+005C
           %x2F /          ; /    solidus         U+002F
           %x62 /          ; b    backspace       U+0008
           %x66 /          ; f    form feed       U+000C
           %x6E /          ; n    line feed       U+000A
           %x72 /          ; r    carriage return U+000D
           %x74 /          ; t    tab             U+0009
           %x75 4HEXDIG )  ; uXXXX                U+XXXX

escape = %x5C              ; \

quotation-mark = %x22      ; "

unescaped = %x20-21 / %x23-5B / %x5D-10FFFF
```

Encoding in JSON: RFC 4627

JSON text SHALL be encoded in Unicode. The default encoding is UTF-8.

Since the first two characters of a JSON text will always be ASCII characters [RFC0020], it is possible to determine whether an octet stream is UTF-8, UTF-16 (BE or LE), or UTF-32 (BE or LE) by looking at the pattern of nulls in the first four octets.

```
00 00 00 xx  UTF-32BE
00 xx 00 xx  UTF-16BE
xx 00 00 00  UTF-32LE
xx 00 xx 00  UTF-16LE
xx xx xx xx  UTF-8
```

MySQL e charset

Support charset di MySQL

- charset = Character encoding: ormai dovrebbe essere chiaro cosa si intende
- Collation: una serie di regole per il confronto fra stringhe (è il motivo dell'esistenza dei charset nei DB)
- Collation più banale: binary collation, confronta le stringhe byte a byte
- La collation (collazione) in generale dipende dal charset
- MySQL è uno dei pochi database che consentono di avere charset diversi a livello di colonna
- La maggior parte dei database definisce il charset e la collation per un intero database
- MySQL è in grado di mescolare stringhe provenienti di diversi charset e diverse collation: questo è possibile grazie a precise regole

```
show variables like "character_set%";
```

Variable_name	Value
character_set_client	latin1
character_set_connection	latin1
character_set_database	latin1
character_set_filesystem	binary
character_set_results	latin1
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/data/mysql/iovino/share/mysql/charsets/

```
show variables like "collation%";
```

Variable_name	Value
collation_connection	latin1_swedish_ci
collation_database	latin1_swedish_ci
collation_server	latin1_swedish_ci

mysql> show character set;

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
euckr	EUC-KR Korean	euckr_korean_ci	2
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
greek	ISO 8859-7 Greek	greek_general_ci	1
cp1250	Windows Central European	cp1250_general_ci	1
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
utf8	UTF-8 Unicode	utf8_general_ci	3
ucs2	UCS-2 Unicode	ucs2_general_ci	2
cp866	DOS Russian	cp866_general_ci	1
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
cp1251	Windows Cyrillic	cp1251_general_ci	1
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
binary	Binary pseudo charset	binary	1
geostd8	GEOSTD8 Georgian	geostd8_general_ci	1
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

- Due diversi charset non possono avere la stessa collation
- Ogni charset ha una default collation
- Convenzione: _ci, _cs (case insensitive/sensitive), _bin (binary)
- http://www.collation-charts.org/
- http://www.unicode.org/reports/tr10/
- Language: Swedish: z < ö, German: ö < z
- Usage: German Dictionary: of < öf, German Telephone: öf <of
- Customizations: Upper-First: A < a, Lower-First: a < A

Unicode support in MySQL

- Versioni >= 4.1, 5.0, 5.1
  - ucs2
  - utf8 (3 byte UTF-8, solo BMP)
- Versioni >= 5.5
  - ucs2
  - utf16
  - utf16le (>= 5.6)
  - utf32
  - utf8 (3 byte UTF-8, solo BMP, in future release potrebbe cambiare, e diventare di 4 byte)
  - utf8mb3 (alias di utf8)
  - utf8mb4 (4 byte UTF-8, tutto Unicode)

Esempio degli effetti della collation

- Nella colonna X (charset latin 1) della tabella T abbiamo i seguenti dati:
  Muffler
  Müller
  MX Systems
  MySQL
- **`SELECT X FROM T ORDER BY X`**

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

Server charset e collation

- Esistono un server charset e una server collation (variabili di sistema di MySQL server)
- Tali variabili sono utilizzate unicamente come charset e collation di default alla creazione di un database (se non esplicitamente dichiarato):
  `CREATE DATABASE`
- Default: charset latin1 e collation latin1_swedish_ci (almeno fino alla v.5.6 compresa)
- Per cambiare questi default senza agire sulla configurazione occorre ricompilare MySQL
- Possono essere impostate nella configurazione (/etc/my.cnf)
- Attenzione: tali impostazioni di default non influenzano il charset di default della connessione, ovvero come il server "pensa" che un client gli parli (**`skip-character-set-client-handshake`**)

Database charset e collation

- **`CREATE DATABASE`** e **`ALTER DATABASE`** hanno clausole opzionali che consentono di specificare il charset e la collation del database
- **`CREATE DATABASE`** ***`db_name`*** **`CHARACTER SET`** ***`X`*** **`COLLATE`** ***`Y`*****`;`**
- Se X e Y sono specificati entrambi (e compatibili) vengono impostati per il database
- Se solo X è specificato, viene utilizzata la collation di default di X
- Se viene specificata solo Y, viene utilizzato il charset corrispondente
- Se nessuno dei due è specificato, vengono utilizzati charset e collation del server
- Il database charset è utilizzato anche dallo statement **`LOAD DATA INFILE`**

Table charset e collation

- **`CREATE TABLE`** e **`ALTER TABLE`** hanno clausole opzionali che consentono di specificare il charset e la collation della tabella
- **`CREATE TABLE`** ***`tbl_name`*** **`(...)`**
  `CHARACTER SET` ***`X`*** **`COLLATE`** ***`Y`*****`;`**
- Se X e Y sono specificati entrambi (e compatibili) vengono impostati per la tabella
- Se solo X è specificato, viene utilizzata la collation di default di X
- Se viene specificata solo Y, viene utilizzato il charset corrispondente
- Se nessuno dei due è specificato, vengono utilizzati charset e collation del database
- Charset e collation della tabella verranno utilizzate per le colonne per le quali non vengono esplicitamente dichiarati

Column charset e collation

- Tutte le colonne di tipo CHAR, VARCHAR e TEXT hanno un charset e una collation di colonna
- Le column definition di **`CREATE TABLE`** e **`ALTER TABLE`** hanno clausole opzionali che consentono di specificare il charset e la collation della colonna
- ```
  CREATE TABLE t1 (
      col1 VARCHAR(5)
           CHARACTER SET latin1
           COLLATE latin1_german1_ci);
  ```
- Valgono le stesse regole viste precedentemente sulla precedenza delle assegnazioni
- Esempio:

```
CREATE TABLE t1 (
    col1 CHAR(10) CHARACTER SET utf8
    ) CHARACTER SET latin1
    COLLATE latin1_bin;
```
```

Literal charset e collation

- I literal sono le stringhe dichiarate direttamente negli statement SQL
- Anch'esse hanno un charset e una collation che possono essere dichiarati tramite un "introducer":
  `SELECT '`*`string`*`';`
  `SELECT _latin1'`*`string`*`';`
  `SELECT _latin1'`*`string`*`' COLLATE latin1_danish_ci;`
- Nel primo statement il charset e la collation sono definiti dalle variabili di sistema **`character_set_connection`** e **`collation_connection`**
- L'introducer non modifica il parsing delle sequenze di escape, che rimane definito dai parametri di connessione

National charset

- L’SQL Standard prevede il tipo di dato **`NCHAR`** (o **`NATIONAL CHAR`**) ad indicare un tipo di colonna CHAR che utilizza un qualche charset di default
- MySQL (>=4.1) utilizza utf8 per questo tipo di dato
- Per esempio i seguenti statement sono equivalenti:
  `CHAR(10) CHARACTER SET utf8 NATIONAL CHARACTER(10)`
  `NCHAR(10)`
- Valgono analoghe considerazioni per i varchar

Esempi di assegnazione charset e collation

- **CREATE TABLE t1 (c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;**
- **CREATE TABLE t1 (c1 CHAR(10) CHARACTER SET latin1) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;**
- **CREATE TABLE t1 (c1 CHAR(10)) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;**
- **CREATE DATABASE d1 DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci; USE d1; CREATE TABLE t1 (c1 CHAR(10));**

Charset e collation della connessione 1/2

- In una normale connessione al database client e server si scambiano query (client) e result sets o messaggi (server)
- Il server utilizza la variabile character_set_client per determinare il charset in cui “parla” il client
- Il server traduce il charset del client nel **`character_set_connection`** e nella **`collation_connection`** (eccetto per le stringhe che hanno un introducer)
- Per la comparazione delle stringhe literal viene utilizzata la **`collation_connection`**
- Per la comparazione delle stringhe con in valori delle colonne, viene utilizzata la collation di colonna
- Il server risponde al client utilizzando la variabile **`character_set_results`**
- Il client può impostare TUTTE queste variabili, o affidarsi al default

Charset e collation della connessione 2/2

- **`SET NAMES 'x';`** equivale a:

```
SET character_set_client = x;
SET character_set_results = x;
SET character_set_connection = x;
```

- **`SET CHARACTER SET 'x';`** equivale a:

```
SET character_set_client = x;
SET character_set_results = x;
SET collation_connection =
@@collation_database;
```

- Attenzione: il default di queste variabili è **latin1**, a meno che non sia stato specificato un diverso default per il database, in combinazione con l'opzione di configurazione **`skip-character-set-client-handshake`**

Charset dei messaggi di errore

- In MySQL 5.0:
  - Il message template usa il charset associato alla lingua del messaggio di errore (latin1 per l'inglese, koi8r per il russo, etc..)
  - I parametri nel messaggio di errore utilizzano altri charset:
    - UTF-8 per gli identificatori di tabella e colonna
    - charset proprio dei dati
- Questo porta a un possibile mix di charset nel messaggio
- Il problema è stato risolto dal MySQL 5.5 in cui i messaggi di errore sono costruiti internamente utilizzando UTF-8 e ritornano al client in **`character_set_results`** con escaping Unicode per i caratteri non rappresentabili in quel charset

Uso di “COLLATE” nei comandi SQL

- Si può effettuare un override della collation di colonna con il predicato **`COLLATE`** nelle clausole **`ORDER BY`**, **`AS`**, **`GROUP BY`**, funzioni di aggregazione, **`DISTINCT`**, **`WHERE`**, **`HAVING`**
- **`SELECT k FROM t1 ORDER BY k COLLATE latin1_german2_ci;`**
- **`SELECT k COLLATE latin1_german2_ci AS k1 FROM t1 ORDER BY k1;`**
- **`SELECT * FROM t1 WHERE k LIKE 'Müller' COLLATE latin1_german_ci`**

Collation delle espressioni SQL

- SELECT x FROM T WHERE x = 'Y';
- Si usa la collation di x o del literal 'Y' ?
- Quando “si mescolano” i dati, MySQL utilizza un sistema di punteggi (coercibility rules):
  - COLLATE : 0
  - Concat di 2 stringhe di diversa collation: 1
  - Colonna, stored parameter, local var: 2
  - System constant: 3
  - Literal: 4
  - NULL: 5
- Si usa la collation con il valore di coercibility più basso
- Se entrambi i lati hanno lo stesso valore, allora:
  - Se entrambi sono o non sono Unicode, è un errore
  - Se uno è Unicode e l’altro no, il lato Unicode vince
    `SELECT CONCAT(utf8_column, latin1_column) FROM t1;`
  - In un mix di collation _bin e _ci o _cs, vince _bin
- column1 = 'A' (collation di column 1)
- column1 COLLATE x (collate x)

_bin collations e binary types

- CHAR, VARCHAR, TEXT (charset e collation)
- BINARY, VARBINARY, BLOB (binary strings)
- In cosa differisce un elemento (colonna, literal, etc.) con collation di tipo “_bin” rispetto a un binary?
  - Unità di sorting e comparazione
  - Conversione di charset
  - Lettercase conversion
  - Gestione dei trailing space nei confronti e in inserimenti e recuperi del tipo CHAR

Esempi (_bin e binary)

```
SET NAMES latin1;
SELECT 'a  ' = 'A';
1
SET NAMES latin1 COLLATE latin1_bin;
SELECT LOWER('aA');
aa
SELECT 'a' = 'A';
0
SELECT 'a' = 'a ';
1
SET NAMES binary
SELECT LOWER('aA');
aA
SELECT 'a ' = 'a';
0
```

L’operatore **BINARY**

- Effettua il casting della stringa che lo segue a stringa binaria: forza la collation a binary
  `SELECT 'a' = 'A'; /* 1 */`
  `SELECT BINARY 'a' = 'A'; /* 0 */`
- Nella definizione delle colonne ha un altro significato, riguardante la collation o il charset:
- **`CHAR(10) BINARY`**
  è equivalente a
  `CHAR(10) CHARACTER SET latin1 COLLATE latin1_bin`
  se il default charset sella tabella è latin 1

  `VARCHAR(10) CHARACTER SET binary`
  è equivalente a
  `VARBINARY(10)`

Lunghezza delle stringhe in MySQL

- La funzione **`LENGTH()`** ritorna la lunghezza dei byte della stringa e non la lunghezza in caratteri: utilizzare **`CHAR_LENGTH()`**
- Le altre funzioni di operazioni sulle stringhe sono binary safe (**`LEFT`**, **`RIGHT`**, **`SUBSTR`**...)

Result Strings

- Il charset e la collation delle stringhe di ritorno da funzioni MySQL sulle stringhe è, in generale, lo stesso dell'input:
  `UPPER(x)` ha il charset e la collation di x
- N.B.: **`REPLACE()`** ignora sempre la collation ed effettua una comparazione case sensitive
- In operazioni che combinano input multipli ed emettono output singoli:
  - Se esiste un esplicito COLLATE X, usa X
  - Se esistono COLLATE X e COLLATE Y, errore
  - Se tutte le collation sono X, usa X
  - Negli altri casi, la collation è indefinita

Convert e Cast

```
CONVERT(expr USING transcoding_name)
INSERT INTO utf8table (utf8column)
  SELECT CONVERT(latin1field USING utf8)
  FROM latin1table;

CAST(character_string AS
  character_data_type CHARACTER SET
  charset_name)
SELECT CAST(_latin1'test' AS CHAR
  CHARACTER SET utf8);
```

- Utilizzando CAST senza specificare il charset verrà utilizzato quello della connessione

UTF-8 per i Metadata

- Tutti i metadati (nomi delle tabelle, delle colonne, etc..) di MySQL sono in Unicode, di fatto in UTF-8
- Le funzioni di ritorno **`USER()`**, **`CURRENT_USER`**, **`SESSION_USER()`**, **`SYSTEM_USER`**, **`DATABASE()`** e **`VERSION()`** hanno charset UTF-8 di default
- La variabile di sistema **`character_set_system`** descrive il charset dei metadati
- Attenzione: i nomi delle colonne sono sempre valutati in **`character_set_client`** e restituiti in **`character_set_results`**

Locale in MySQL

- Le impostazioni di localizzazione in MySQL riguardano la lingua utilizzata per i nomi e abbrevazioni dei giorni della settimana e dei mesi
- **`SELECT @@lc_time_names;`**
- Investe le funzioni **`DATE_FORMAT()`**, **`DAYNAME()`**, **`MONTHNAME()`**
- Utilizza language tag e region subtag (default en_US)
- Esempio:
  `SELECT DAYNAME(NOW());`
  Wednesday
  `SET lc_time_names = 'it_IT';`
  `SELECT DAYNAME(NOW());`
  mercoledì

Time zone in MySQL

- Time zone di sistema
- Time zone server
  `SET GLOBAL time_zone = timezone;`
  `SELECT @@global.time_zone;`
- Per connection time zone
  `SET time_zone = timezone;`
  `SELECT @@session.time_zone;`
- Il valore può essere impostato
  - **`SYSTEM`**
  - Offset da UTC **`'+10'`**, **`'-6:00'`**
  - Named time zone **`'Europe/Rome'`**, **`'CET'`** (se popolate all'installazione)