3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

1: Why do we need both UCS and Unicode character sets? - Stack Overflow

UCS-4 uses twice as much memory than UCS-2, but it supports all Unicode characters. UTF is a compromise between UCS-2 and UCS characters in the BMP range use one UTF unit (16 bits), characters outside this range use two UTF units (a surrogate pair, 32 bits).

UCS contains all characters of all other character set standards. It also guarantees round-trip compatibility, i. UCS contains the characters required to represent practically all known languages. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This might eventually include not only Hieroglyphs and various historic Indo-European languages, but even some selected artistic scripts such as Tengwar, Cirth, and Klingon. The UCS standard ISO describes a bit character set architecture consisting of bit groups, each divided into bit planes made up of 8-bit rows with column positions, one for each character. Part 2 of the standard ISO adds characters to group 0 outside the BMP in several supplementary planes in the range 0x to 0x10ffff. There are no plans to add characters beyond 0x10ffff to the standard, therefore of the entire code space, only a small fraction of group 0 will ever be actually used in the foreseeable future. The BMP contains all characters found in the commonly used other character sets. The supplemental planes added by ISO cover only more exotic characters for special scientific, dictionary printing, publishing industry, higher-level protocol and enthusiast needs. Combining characters Some code points in UCS have been assigned to combining characters. These are similar to the nonspacing accent keys on a typewriter. A combining character just adds an accent to the previous character. The most important accented characters have codes of their own in UCS, however, the combining character mechanism allows us to add accents and other diacritical marks to any character. The combining characters always follow the character which they modify. For example, the German character Umlaut-A "Latin capital letter A with diaeresis" can either be represented by the precomposed UCS code 0x00c4, or alternatively as the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": Combining characters are essential for instance for encoding the Thai script or for mathematical typesetting and users of the International Phonetic Alphabet. Implementation levels As not all systems are expected to support advanced mechanisms like combining characters, ISO specifies the following three implementation levels of UCS: Level 2 In addition to level 1, combining characters are now allowed for some languages where they are essential e. Level 3 All UCS characters are supported. The Unicode standard and technical reports published by the Unicode Consortium provide much additional information on the semantics and recommended usages of various characters. They provide guidelines and algorithms for editing, sorting, comparing, normalizing, converting and displaying Unicode strings. To signal the use of UTF-8 as the character encoding to all applications, a suitable locale has to be selected via environment variables e. Under Linux, in general only the BMP at implementation level 1 should be used at the moment. Up to two combining characters per base character for certain scripts in particular Thai are also supported by some UTF-8 terminal emulators and ISO fonts level 2 , but in general precomposed characters should be preferred where available Unicode calls this Normalization Form C. Private area In the BMP, the range 0xe to 0xf8ff will never be assigned to any characters by the standard and is reserved for private usage. For the Linux community, this private area has been subdivided further into the range 0xe to 0xefff which can be used individually by any end-user and the Linux zone in the range 0xf to 0xf8ff where extensions are coordinated among all Linux users. The registry of the characters assigned to the Linux zone is currently maintained by H. Architecture and Basic Multilingual Plane. This is the official specification of UCS. A good reference book about the C programming language. The fourth edition covers the Amendment 1 to the ISO C90 standard, which adds a large number of new C library functions for handling wide and multibyte character encodings, but it does not yet cover ISO C99, which improved wide and multibyte character support even further. Provides subscription information for the linux-utf8 mailing list, which is the best place to look for advice on using Unicode under Linux. Bugs When this man page was last revised, the GNU C Library support for UTF-8 locales was mature and XFree86 support was in an advanced state, but work on making applications most notably editors suitable for use in

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

UTF-8 locales was still fully in progress. Current general UCS support under Linux usually provides for CJK double-width characters and sometimes even simple overstriking combining characters, but usually does not include support for scripts with right-to-left writing direction or ligature substitution requirements such as Hebrew, Arabic, or the Indic scripts. These scripts are currently only supported in certain GUI applications HTML viewers, word processors with sophisticated text rendering engines.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

2: utf 8 - What's the difference between Unicode and UTF-8? - Stack Overflow

The Universal Character Set (UCS) is not a standard but something defined in a standard, namely ISO This should not be confused with encodings, such as UCS It is difficult to guess whether you actually mean different encodings or different standards.

This would have been been theoretically correct if UTF would not exist and Unicode would have received more than 1,, characters. Which One To Pick? Out of all the choices UTF-8 immediately looks like the best one. UTF-8 is also not byte order dependent which is an immediate win, but it also works with C strings so is backwards compatible and worst case it only wastes as much memory as all the other formats. Upon further introspection it however becomes clear that depending on the language of the text stored, UTF will become more space efficient. For instance for Japanese text UTF will be more space efficient than UTF-8 as many characters are on the basic plane and with that will only require a single UTF code point whereas they are high enough up in the range that they will require 3 bytes for UTF For text UTF-8 has clearly won. Internal Encodings However the question is not so easy when working with Unicode internally and there have been different opinions on this issue. The nice property of UTF is that it allows you to be sloppy as the vast majority of data you will be presented with is probably in the basic plane. This means that operations like strlen will both return the number of code units as well as the number of characters. For a really long time there did not seem to be much of a contest to using UTF as internal encoding. For a long time the only programming language besides lots of C code that used UTF-8 as internal encoding seemed to be Perl. While Ruby can work with lots of internal string encodings, UTF-8 is the one you find most commonly. That would actually be fine, if programming languages would provide a data type with at least 21 bit of precision to hold a whole Unicode character though. C and Java unfortunately do not support that. That Java does not provide it makes sense to some degree considering the age of the language and how the string is exposed. That C does not support it is unfortunate however. Rust and Go for instance have this better sorted out. While they do use UTF-8 as internal string encoding and expose this to the user, they provide 32 bit data types called rune in Go and char in rust. In both programming languages you can iterate in actual Unicode characters over the whole string. In many cases this is plenty because parsing for instance usually only needs to look at one or two characters at the time. In many ways the question is how valuable constant time addressing of a single character in strings is. What Rust and Go gain from having UTF-8 strings is that they are very efficient when they need to juggle with bytes next to textual content. To take Rust as an example, parsing protocols is very efficient because in many cases a parsing step becomes a simple memcpy. The reason for this is that so much data out there is UTF After copying of the data you just need to do a basic check afterwards if the UTF-8 is not invalid, which can be nicely optimized. In contrast to that UTF is a trickier because you need to figure out the length of the buffer through an initial scan and then a second one to decode the data. Or you do it in one go and overallocate. Go even gets away with using completely unchecked UTF-8 strings. Go on the other hand lets you happily mix random bytes into your string, but all IO operations are required to ensure that the data is valid. For a really long time it looked like nobody would challenge the idea of using UTF as internal string format but that seems to change now. On one hand some languages are exploring using UTF-8 internally, on the other hand we have Python 3 which explores the idea of switching between latin1, UCS-2 and UTF on a string-by-string basis. The Python 3 trick sounded quite good on the paper but I noticed that there are some practical downsides. For instance Emojis are outside of the basic plane which means that Python needs to represent them as UTF internally. With how lots of template engines are currently structured that can cause some interesting characteristics. Jinja2 for instance currently renders in Unicode and then has a separate encoding step. If you would build a github comment page and an Emoji would be in the comments then whole your response upgrades to UTF just because of a single character. In corner cases like this it might be interesting to use the streaming interface of Jinja2 to encode chunk by chunk to UTF-8 to avoid the extra cost of a more expensive internal string. Having worked with Rust for a while now I am getting more and more convinced that the approach is a good idea. While it forces you to give up on the idea of being able to address

characters individually, that is actually not a huge loss. For a start Unicode would pretty much require you to normalize your strings anyways before you do text processing due to the many ways in which you can format the strings. For instance umlauts come in combined characters but they can also be manually created by placing the regular letter followed by the combining diaeresis character. So for quite a few operations like validating length, font rendering etc. Especially if UTF-8 stays the dominant format then keeping it internally as well makes a lot of sense and requires lots of unnecessary encoding and decoding steps and means the language does not need to provide support for ASCII strings separately.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

3: UTF - Wikipedia

The Universal Coded Character Set (UCS) is a standard set of characters defined by the International Standard ISO/IEC , Information technology â€" Universal Coded Character Set (UCS) (plus amendments to that standard), which is the basis of many character encodings.

UCS contains all characters of all other character set standards. It also guarantees round-trip compatibility, i. UCS contains the characters required to represent practically all known languages. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This might eventually include not only Hieroglyphs and various historic Indo-European languages, but even some selected artistic scripts such as Tengwar, Cirth, and Klingon. The UCS standard ISO describes a bit character set architecture consisting of bit groups, each divided into bit planes made up of 8-bit rows with column positions, one for each character. Part 2 of the standard ISO adds characters to group 0 outside the BMP in several supplementary planes in the range 0x to 0x10ffff. There are no plans to add characters beyond 0x10ffff to the standard, therefore of the entire code space, only a small fraction of group 0 will ever be actually used in the foreseeable future. The BMP contains all characters found in the commonly used other character sets. The supplemental planes added by ISO cover only more exotic characters for special scientific, dictionary printing, publishing industry, higher-level protocol and enthusiast needs. Combining characters Some code points in UCS have been assigned to combining characters. These are similar to the nonspacing accent keys on a typewriter. A combining character just adds an accent to the previous character. The most important accented characters have codes of their own in UCS, however, the combining character mechanism allows us to add accents and other diacritical marks to any character. The combining characters always follow the character which they modify. For example, the German character Umlaut-A "Latin capital letter A with diaeresis" can either be represented by the precomposed UCS code 0x00c4, or alternatively as the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": Combining characters are essential for instance for encoding the Thai script or for mathematical typesetting and users of the International Phonetic Alphabet. Implementation levels As not all systems are expected to support advanced mechanisms like combining characters, ISO specifies the following three implementation levels of UCS: Level 2 In addition to level 1, combining characters are now allowed for some languages where they are essential e. Level 3 All UCS characters are supported. The Unicode standard and technical reports published by the Unicode Consortium provide much additional information on the semantics and recommended usages of various characters. They provide guidelines and algorithms for editing, sorting, comparing, normalizing, converting and displaying Unicode strings. To signal the use of UTF-8 as the character encoding to all applications, a suitable locale has to be selected via environment variables e. Under Linux, in general only the BMP at implementation level 1 should be used at the moment. Up to two combining characters per base character for certain scripts in particular Thai are also supported by some UTF-8 terminal emulators and ISO fonts level 2 , but in general precomposed characters should be preferred where available Unicode calls this Normalization Form C. Private area In the BMP, the range 0xe to 0xf8ff will never be assigned to any characters by the standard and is reserved for private usage. For the Linux community, this private area has been subdivided further into the range 0xe to 0xefff which can be used individually by any end-user and the Linux zone in the range 0xf to 0xf8ff where extensions are coordinated among all Linux users. The registry of the characters assigned to the Linux zone is currently maintained by H. Architecture and Basic Multilingual Plane. This is the official specification of UCS. A good reference book about the C programming language. The fourth edition covers the Amendment 1 to the ISO C90 standard, which adds a large number of new C library functions for handling wide and multibyte character encodings, but it does not yet cover ISO C99, which improved wide and multibyte character support even further. Provides subscription information for the linux-utf8 mailing list, which is the best place to look for advice on using Unicode under Linux. Current general UCS support under Linux usually provides for CJK double-width characters and sometimes even simple overstriking combining characters, but usually does not include support for scripts with right-to-left

writing direction or ligature substitution requirements such as Hebrew, Arabic, or the Indic scripts. These scripts are currently supported only in certain GUI applications HTML viewers, word processors with sophisticated text rendering engines. A description of the project, and information about reporting bugs, can be found at http:

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

4: UCS-2 and UTF-8

The UCS standard (ISO) describes a bit character set architecture consisting of bit groups, each divided into bit planes made up of 8-bit rows with column positions, one for each character.

This makes the term too imprecise to use when specifying algorithms, protocols, or document formats, unless you explicitly define what you mean by it. It is particularly important to remember that bytes only rarely equate to characters in Unicode, as shown in the earlier examples. However, particularly in complex scripts, what a user perceives as a smallest component of their alphabet and so what we will call a user-perceived character may actually be a sequence of code points. It is often important to take into account these user-perceived characters. For example, it is common to treat certain combinations of code points as a single unit for various editing operations, such as line-breaking, cursor movement, selection, deletion, etc. It would usually be problematic if a user selection accidentally omitted part of the letters just mentioned, or if a line-break separated a base character from its following combining characters. In order to approximate user-perceived character units for such operations, Unicode uses a set of generalised rules to define grapheme clusters â€" sequences of adjacent code points that can be treated as a unit by applications. Unicode Standard Annex Text Segmentation actually defines two types of grapheme cluster: It is not recommended to use the latter. Applications that need to work with user-perceived characters in Bangla therefore need to apply some script-specific tailoring of the grapheme cluster rules. CSS, in order to refer to an indivisible text unit in a given context, uses the term typographic character unit. The definition of what constitutes a typographic character unit depends on the operation that is being applied. The determination of what constitutes a typographic character unit in a given language and editing context is deferred to the application, rather than spelled out in rules. A font is a collection of glyphs. In a simple scenario, a glyph is the visual representation of a code point. The glyph used to represent a code point will vary with the font used, and whether the font is bold, italic, etc. In the case of emoji, the glyphs used will vary by platform. In fact, more than one glyph may be used to represent a single code point, and multiple code points may be represented by a single glyph. Emoji provide another example of the complex relationship between code points and glyphs. It can also be formed by using a sequence of code points: Altering or adding other emoji characters can alter the composition of the family. Many common emoji can only be formed using sequences of code points, but should be treated as a single user-perceived character when displaying or processing the text. Character escapes A character escape is a way of representing a character without actually using the character itself. Because the document character set is Unicode, the user agent should recognize that this represents a Hebrew aleph character.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

5: Universal Coded Character Set - Wikipedia

ISO UCS-2 (Unicode) Universal Coded Character Set (UCS) is the name of the ISO standard that defines a single code for the representation, interchange, processing, storage, entry, and presentation of the written form of all the major languages of the world.

White cells are the leading bytes for a sequence of multiple bytes, the length shown at the left edge of the row. The text shows the Unicode blocks encoded by sequences starting with this byte, and the hexadecimal code point shown in the cell is the lowest character value encoded using that leading byte. Red cells must never appear in a valid UTF-8 sequence. The first two red cells C0 and C1 could be used only for a two-byte encoding of a 7-bit ASCII character which should be encoded in one byte; as described below such "overlong" sequences are disallowed. Pink cells are the leading bytes for a sequence of multiple bytes, of which some, but not all, possible continuation sequences are valid. E0 and F0 could start overlong encodings, in this case the lowest non-overlong-encoded code point is shown. Overlong encodings[edit] In principle, it would be possible to inflate the number of bytes in an encoding by padding the code point with leading 0s. This is called an overlong encoding. The standard specifies that the correct encoding of a code point use only the minimum number of bytes required to hold the significant bits of the code point. Longer encodings are called overlong and are not valid UTF-8 representations of the code point. This rule maintains a one-to-one correspondence between code points and their valid encodings, so that there is a unique valid encoding for each code point. This ensures that string comparisons and searches are well-defined. This allows the byte 00 to be used as a string terminator. Invalid byte sequences[edit] Not all sequences of bytes are valid UTF A UTF-8 decoder should be prepared for: This guarantees that it will neither interpret nor emit an ill-formed code unit sequence. Early versions of Python 3. The inability to deal with UTF-8 without first confirming it was valid actually greatly impeded adoption of Unicode. Modern practice is to replace errors with a replacement character, and to insure that systems do not interpret these replacement characters in any dangerous way. The errors can be detected later when it is convenient to report an error, or display as blocks when the string is drawn for the user. Replacement requires defining how many bytes are in the error. Early decoders would often use the same number of bytes as the lead byte indicated as the length of the error. This had the unfortunate problem that a dropped byte would cause the error to consume some of the next character s. It also was difficult to parse in a reverse direction. In these decoders 0xE0,0x80,0x80 is three errors, not one. This means an error is no more than three bytes long and never contains the start of a valid character. Another popular practice is to turn each byte into an error. In this case 0xE1,0xA0,0xC0 is three errors, not two. The primary advantage is that there are now only different error bytes. This allows the decoder to define different error replacements such as: But this runs into a practical difficulty in that the encoder must make sure the sequence of "errors" it is encoding do not actually turn into valid UTF In addition making any surrogate halves invalid means you cannot encode invalid UTF The Unicode code point for the character represented by the byte in CP ,[citation needed] which is similar to using ISO, except that some characters in the range 0x80â€”0x9F are mapped into different Unicode code points. This makes text where legacy encodings are mixed with UTF-8 readable, and thus it is commonly done in browsers. The large number of invalid byte sequences provides the advantage of making it easy to have a program accept both UTF-8 and legacy encodings such as ISO Software can check for UTF-8 correctness, and if that fails assume the input to be in the legacy encoding. To preserve these invalid UTF sequences, their corresponding UTF-8 encodings are sometimes allowed by implementations despite the above rule. This spelling is used in all the Unicode Consortium documents relating to the encoding.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

6: Programming with Unicode â€" Programming with Unicode

UCS-4 uses twice as much memory than UCS-2, but it supports all Unicode characters. UTF is a compromise between UCS-2 and UCS characters in the BMP range use one UTF unit (16 bits), characters outside this range.

UCS contains all characters of all other character set standards. It also guarantees "round- trip compatibility"; in other words, conversion tables can be built such that no information is lost when a string is converted from any other encoding to UCS and back. UCS contains the characters required to represent practically all known languages. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This might eventually include not only Hieroglyphs and various historic Indo-European languages, but even some selected artistic scripts such as Tengwar, Cirth, and Klingon. The UCS standard ISO describes a bit character set architecture consisting of bit groups, each divided into bit planes made up of 8-bit rows with column positions, one for each character. Part 2 of the standard ISO adds characters to group 0 outside the BMP in several supplementary planes in the range 0x to 0x10ffff. There are no plans to add characters beyond 0x10ffff to the standard, therefore of the entire code space, only a small fraction of group 0 will ever be actually used in the foreseeable future. The BMP contains all characters found in the commonly used other character sets. The supplemental planes added by ISO cover only more exotic characters for special scientific, dictionary printing, publishing industry, higher-level protocol and enthusiast needs. Combining characters Some code points in UCS have been assigned to combining characters. These are similar to the nonspacing accent keys on a typewriter. A combining character just adds an accent to the previous character. The most important accented characters have codes of their own in UCS, however, the combining character mechanism allows us to add accents and other diacritical marks to any character. The combining characters always follow the character which they modify. For example, the German character Umlaut-A "Latin capital letter A with diaeresis" can either be represented by the precomposed UCS code 0x00c4, or alternatively as the combination of a normal "Latin capital letter A" followed by a "combining diaeresis": Combining characters are essential for instance for encoding the Thai script or for mathematical typesetting and users of the International Phonetic Alphabet. Implementation levels As not all systems are expected to support advanced mechanisms like combining characters, ISO specifies the following three implementation levels of UCS: Level 2 In addition to level 1, combining characters are now allowed for some languages where they are essential e. Level 3 All UCS characters are supported. The Unicode standard and technical reports published by the Unicode Consortium provide much additional information on the semantics and recommended usages of various characters. They provide guidelines and algorithms for editing, sorting, comparing, normalizing, converting, and displaying Unicode strings. To signal the use of UTF-8 as the character encoding to all applications, a suitable locale has to be selected via environment variables e. Private area In the Basic Multilingual Plane, the range 0xe to 0xf8ff will never be assigned to any characters by the standard and is reserved for private usage. For the Linux community, this private area has been subdivided further into the range 0xe to 0xefff which can be used individually by any end-user and the Linux zone in the range 0xf to 0xf8ff where extensions are coordinated among all Linux users. Architecture and Basic Multilingual Plane. This is the official specification of UCS. A good reference book about the C programming language. The fourth edition covers the Amendment 1 to the ISO C90 standard, which adds a large number of new C library functions for handling wide and multibyte character encodings, but it does not yet cover ISO C99, which improved wide and multibyte character support even further. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http: Ubuntu and Canonical are registered trademarks of Canonical Ltd.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

7: THE UNIVERSAL CHARACTER SET (UCS)

Unencodable characters When a character string is encoded to a character set smaller than the Unicode character set (UCS), a character may not be encodable. For example, â‚¬ (U+20AC) is not encodable to ISO , but it is encodable to ISO and UTF

EUC similar to the DBCS Shift encodings, with the application of different numeric shift rules, and the introduction of single-shift bytes: Examples of compressing Character Encoding Schemes: BOCU-1 maps each input string to a unique compressed string, but does not map each code unit to a unique series of bytes. Because of compression heuristics, the same input string may result in different byte sequences, but the schemes are fully reversible. Little Endian architectures put the least significant byte at the lower address, while Big Endian architectures start with the most significant byte. This difference does not matter for operations on code units in memory, but the byte order becomes important when code units are serialized to sequences of bytes using a particular CES. In terms of reading a data stream, there are two types of byte order: Same as or Opposite of the byte order of the processor reading the data. In the former case, no special operation needs to be taken; in the latter case, the data needs to be byte reversed before processing. In terms of external designation of data streams, three types of byte orders can be distinguished: At the head of a data stream, the presence of a byte order mark can therefore be used to unambiguously signal the byte order of the code units. In that case, the abstract character repertoire for the character map is the union of the repertoires covered by the coded character sets involved. Unicode Technical Report The text also contains a detailed discussion of issues in mapping between character sets. From the IANA charset point of view it is important that a sequence of encoded characters be unambiguously mapped onto a sequence of bytes by the charset. The charset must be specified in all instances, as in Internet protocols, where textual content is treated as an ordered sequence of bytes, and where the textual content must be reconstructible from that sequence of bytes. In many cases, the same name is used for both a character map and for a character encoding scheme, such as UTFBE. Typically this is done for simple character mappings when such usage is clear from context. Examples include base64, uuencode, BinHex, and quoted-printable. While data transfer protocols often incorporate data compressions to minimize the number of bits to be passed down a communication channel, compression is usually handled outside the TES, for example by protocols such as pkzip, gzip, or winzip. These are data width specifications which are relevant to mail protocols and which appear to predate true TESs like quoted-printable. The Java model supports portable programs, but external data in other encoding forms must first be converted to UTF These two character sets do not have to be the same, but the repertoire of the larger set must include the smaller set to allow mapping from one data type into the other. Other API s supporting UTF are often simply defined in terms of arrays of bit unsigned integers, but this makes certain features of the programming language unavailable, such as string literals. When character data types are passed as arguments in APIs, the byte order of the platform is generally not relevant for code units. The same API can be compiled on platforms with any byte polarity, and will simply expect character data as for any integral-based data to be passed to the API in the byte polarity for that platform. However, the size of the data type must correspond to the size of the code unit, or the results can be unpredictable, as when a byte oriented strcpy is used on UTF data which may contain embedded NUL bytes. While there are many API functions that are designed not to care about which character set the code units correspond to strlen or strcpy for example , many other operations require information about the character and its properties. Thus a Unicode 8-bit string is a sequence of 8-bit Unicode code units; a Unicode bit string is a sequence of bit code units; a Unicode bit string is a sequence of bit code units. Depending on the programming environment, a Unicode string may or may not also be required to be in the corresponding Unicode encoding form. In normal processing, there are many times where a string may be in a transient state that is not well-formed UTF Because strings are such a fundamental component of every program, it can be far more efficient to postpone checking for well formedness. However, whenever strings are specified to be in a particular Unicode encodingâ€”even one with the same code unit sizeâ€”the string must not violate the requirements of that encoding form. For example,

isolated surrogates in a Unicode bit string are not allowed when that string is specified to be well-formed UTF

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

8: Programming With Unicode | Open Library

Besides Unicode that everybody knows, there is also the Universal Character Set as defined by ISO Nowadays Unicode and ISO's UCS are for most practical purposes the same thing with a slightly different name.

There are 3 different implementations of character strings: This advantage is also the main disadvantage of this kind of character string. The length of a character string implemented using UTF is the number of UTF units, and not the number of characters, which is confusing. If the character string only contains characters of the BMP range, the length is the number of characters. Windows 95 uses UCS-2 strings. It is implemented as an array of 8 bits unsigned integers. It can be called by its encoding. The character range supported by a byte string depends on its encoding, because an encoding is associated with a charset. The encoding is not stored explicitly in a byte string. If the encoding is not documented or attached to the byte string, the encoding has to be guessed , which is a difficult task. If a byte string is decoded from the wrong encoding, it will not be displayed correctly, leading to a well known issue: The same problem occurs if two byte strings encoded to different encodings are concatenated. Never concatenate byte strings encoded to different encodings! Use character strings, instead of byte strings, to avoid mojibake issues. PHP5 only supports byte strings. But a UTF-8 string is not a Unicode string because the string unit is byte and not character: Another difference between UTF-8 strings and Unicode strings is the complexity of getting the nth character: There is one exception: An encoding is always associated with a charset. For example, the UTF-8 encoding is associated with the Unicode charset. So we can say that an encoding encodes characters to bytes and decode bytes to characters, or more generally, it encodes a character string to a byte string and decodes a byte string to a character string. The 7 and 8 bits charsets have the simplest encoding: Since these charsets are also called encodings, it is easy to confuse them. The best example is the ISO encoding: Charsets with more than entries cannot encode all code points into a single byte. The encoding encodes all code points into byte sequences of the same length or of variable length. For example, UTF-8 is a variable length encoding: The UCS-2 encoding encodes all code points into sequences of two bytes 16 bits. By default, most libraries are strict: Some libraries allow to choose how to handle them. Most encodings are stateless, but some encoding requires a stateful encoder. By default, most libraries raise an error if a byte sequence cannot be decoded. Most encodings are stateless, but some encoding requires a stateful decoder.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

9: UTF-8 - Wikipedia

Programming with UnicodeÂ¶. 1. About this book. License; Thanks to; Notations; 2. Unicode nightmare.

The simplest, UCS-2, [Note 1] uses a single code value defined as one or more numbers representing a code point between 0 and 65, for each character, and allows exactly two bytes one bit word to represent that value. UCS-2 thereby permits a binary representation of every code point in the BMP, as long as the code point represents a character. Unicode also adopted UTF, but in Unicode terminology, the high-half zone elements become "high surrogates" and the low-half zone elements become "low surrogates". UCS-4 allows representation of each value as exactly four bytes one bit word. As in UCS-2, every encoded character has a fixed length in bytes, which makes it simple to manipulate, but of course it requires twice as much storage as UCS More than half of all Web pages are encoded in UTF It is also increasingly being used as the default character encoding in operating systems, programming languages, APIs, and software applications. Hugh McGregor Ross was one of its principal architects. That standard differed markedly from the current one. The Latin capital letter A, for example, had a location in group 0x20, plane 0x20, row 0x20, cell 0x One could code the characters of this primordial ISO standard in one of three ways: UCS-4, four bytes for every character, enabling the simple encoding of all characters; UCS-2, two bytes for every character, enabling the encoding of the first plane, 0x20, the Basic Multilingual Plane, containing the first 36, codepoints, straightforwardly, and other planes and groups by switching to them with ISO escape sequences; UTF-1 , which encodes all the characters in sequences of bytes of varying length 1 to 5 bytes, each of which contain no control codes. In , therefore, two initiatives for a universal character set existed: Unicode , with 16 bits for every character 65, possible characters , and ISO The software companies refused to accept the complexity and size requirement of the ISO standard and were able to convince a number of ISO National Bodies to vote against it. Two changes took place: Meanwhile, in the passage of time, the situation changed in the Unicode standard itself: For that reason, ISO was limited to contain as many characters as could be encoded by UTF and no more, that is, a little over a million characters instead of over million. Rob Pike and Ken Thompson , the designers of the Plan 9 operating system, devised a new, fast and well-designed mixed-width encoding, which came to be called UTF-8 , [1] currently the most popular UCS encoding. Differences from Unicode[edit] ISO and Unicode have an identical repertoire and numbersâ€”the same characters with the same numbers exist on both standards, although Unicode releases new versions and adds new characters more often. Unicode has rules and specifications outside the scope of ISO In contrast, Unicode adds rules for collation , normalization of forms , and the bidirectional algorithm for right-to-left scripts such as Arabic and Hebrew. For interoperability between platforms, especially if bidirectional scripts are used, it is not enough to support ISO ; Unicode must be implemented. Unicode intends these properties to support interoperable text handling with a mixture of languages. Some applications support ISO characters but do not fully support Unicode. One such application, Xterm , can properly display all ISO characters that have a one-to-one character-to-glyph mapping[clarification needed] and a single directionality. It can handle some combining marks by simple overstriking methods, but cannot display Hebrew bidirectional , Devanagari one character to many glyphs or Arabic both features. Most GUI applications use standard OS text drawing routines which handle such scripts, although the applications themselves still do not always handle them correctly. And even though it is a separate standard, the term Unicode is used just as often, informally, when discussing the UCS. The repertoire, character names, and code points of Unicode Version 2.

3.12 UNICODE: AN UNIVERSAL CHARACTER SET (UCS 8 pdf

Cover letter magic The monuments of Upper Egypt An informal education Accu chek performa manual espaÃ±ol Parties and their principles Russian adult humor That evening sun Internal medicine and the structures of modern medical science The production of iron and steel in Canada during the calendar year 1913 Nts book 2016 Counselling women in violent relationships Contemporary Turkish Politics Managing Hotels Effectively Inst Man I-Net Exam Cram (Exam: 1KO-001) John McCain John B. Judis Cats I have known and loved How To Handle Disillusionment Book of acid Explaining aggregation in Thailand A naturalist in western China Wisdom Of The Generals Washboard weepers: women writers, women listeners, and the debate over soap operas 2007 nissan versa repair manual New-Englands plantation (facsim. of 3rd ed.) Homeschooling for Success The Case of bilingual education strategies Young peoples dictionary Perfect peace by daniel black Catastrophe in Czechoslovakia An American Art Colony Skating Superstars Oral tradition and the history of segmentary societies I stage my first death scene. How Elephants Lost Their Wings Some place quite unknown The life of Michelangelo. Soren kierkegaard either or The other Bishop Berkeley Handbook of software engineering Papillons 2007 Calendar