Bits & Bytes

Arkansas' Premier Computer Club

July 2024

The Bella Vista Computer Club - John Ruehle Center

Highlands Crossing Center, 1801 Forest Hills Blvd Suite 208 (lower level), Bella Vista, AR 72715

Website: http://BVComputerClub.org **Email: editor@bvcomputerclub.org**

MEETINGS

Board Meeting: July 8, 6pm, in John Ruehle Training Center, Highlands Crossing Center.

General Meeting: July 8, 7pm. Program: "Should I Allow My Browser To Manage My Passwords?, with Joel Ewing.

We will meet in-person in John Ruehle Training Center, Highlands Crossing Center, lower level, 1801 Forest Hills Blvd, Bella Vista, or you may attend the meeting on-line via Zoom. Zoom access information is published on our website.

Visitors or Guests are welcome.

Consider attending by Zoom if you are unable to attend in-person.

HELP CLINICS

July 6, 9am - noon at John Ruehle center
July 17, 9am - noon at John Ruehle center
Members may request Remote Help on our website at https://bvcomputerclub.org at menu path Member Benefits ►Remote Help .

MEMBERSHIP

Single membership is $30; $15 for each additional family member in the same household.

Join on our website at https://bvcomputerclub.org at menu path Get Involved ►Join/Renew, by mailing an application (from the web site) with check, or complete an application and pay in person at any meeting.

CLASSES

(At BVCC Training Center)

A 4 hours class in two parts: Tuesday, July 16, 9am-11am (Pt 1), and Thursday, July 18 ,9am-11am (Pt 2), "Microsoft Excel", with Joel Ewing.

Advance sign up required for each listed class: For reservations: email to edu@bvcomputerclub.org, or sign up at the General Meeting. Classes are **free to Computer Club members.**

Check the monthly calendar and announcements for any last minute schedule changes at https://bvcomputerclub.org .

NEW OR RETURNING BVCC MEMBERS

We are pleased to welcome the following new members or members returning as BVCC members after an absence:

David Kleeb Pat Cox Connie Cain

THE ROAD TO UNICODE

By Joel Ewing, President, Bella Vista Computer Club
Bits & Bytes, July 2024
https://bvcomputerclub.org
president (at) bvcomputerclub.org

Background

A "glyph" is a symbolic figure or a character. The most commonly used glyphs in English are the upper- and lower-case Latin letters, A-Z, a-z, the Arabic digits 0-9, a variety of punctuation marks, and various special symbols such as the percent sign used in business. A "font" is a collection of glyphs of a specific style and size.

In documents prepared by hand, the number of glyphs and fonts used in a document were only restricted by the skill of the writer. When mechanized ways of printing became common in the 15th century and beyond, the maximum number of supported glyphs and fonts was restricted by economics and the technology limits of the time. People of my age who learned how to use a typewriter were accustomed to being limited to two fonts, Pica and Elite, and a maximum of about 80 distinct characters. In certain types of documents, it was not uncommon to have to write in some special symbols by hand. To efficiently deal with foreign languages requiring a different alphabet required typewriters with a font customized for that language.

Electro-mechanical teleprinters developed in the late 19th century to automate telegraphy were restricted to upper case letters and digits. Although very slow by today's standards, their successors were used by some early digital computers as a keyboard input device, as a printing device for output, and for their ability as a storage device to punch and read paper tape.

In the late 1940s IBM began to produce punched-card based accounting machines that could do "high-speed" printing of up to 18,000 characters / minute. This speed, which was revolutionary at the time, was achieved by printing an entire line of 120 characters in parallel with a separate printing hammer for each column, at a speed of up to 150 lines per minute, and was made feasible by limiting the device to 48 unique characters: the uppercase characters, digits, and 12 special characters considered essential to business at the time. That "line printer" became the basis for line printers used on IBM's first commercial digital computers in the 1950s, and lack of practical dual case printers was no doubt one of the reasons early digital computers only supported uppercase letters. By the late 1960s dual case print support was available, but at a significant penalty in both print speed and quality. Businesses only used dual case for high-volume, computer-printed documents when it could be cost-justified, which in some cases wasn't until cheap PC-based laser printers became practical as mainframe printers in the 21st century.

Digital Computers and Character Codesets

Digital computers only work with numeric values internally, so how can they deal with alphabet characters and other symbols? Lady Ada Lovelace, a mathematician who worked with Charles Babbage's Analytical Engine design in the 19th century 100 years before the construction of practical general purpose digital computers, explained how. Computers would work with non numeric characters and symbols by using numbers to represent them.

A character codeset allows a computer to represent text by defining a specific mapping between numeric values and specific symbols or glyphs. In the early days of digital computers, it was not unusual for different computer manufacturers to to create their own unique character codeset, or to even have different codesets used by different computers from the same manufacturer. This made sharing data between different types of compute systems a problem.

Computers that stored values as decimal digits would use two digits to represent a character or symbol, giving a possible maximum of 100 different characters; although not all values were defined. Computers that stored values as binary digits initially used 6 bits, representing at most 64 unique characters or symbols. Depending on the context in which that data was used, it was also necessary to assign code values to a "space", and to various control functions like tab, newline, backspace, etc. A limit of 64 values was insufficient to represent both upper and lowercase letters, numbers, and the special characters need by business, so only uppercase letters were supported.

By 1970 it was clear that digital computers using binary-based storage were more cost-effective than decimal-based designs, and also that 8 bits should be the minimum used to represent characters, providing up to 256 unique values. Unfortunately, by then the computing world had evolved into two competing families of codes: those based on IBM's EBCDIC[1] code that had evolved from their punched-card technology and was widely used on IBM mainframes, and the 8-bit ANSI[2] codes that contained the 7-bit ASCII[3] code standard as its first 128 characters and was used on most other computer platforms such as Unix, and eventually on Linux, and Windows. Both of these code families had many variants in order to support foreign language requirements and special symbols that weren't included by default, creating a hodge podge of incompatible codes that complicated storage, processing, and display of character data in a global economy.

The issue of codeset confusion couldn't be resolved within the constraints of 8-bit character codes. It would be two decades before hardware costs and processing speeds attained levels where serious consideration would be give to the implementation of larger codesets as a solution.

Unicode To The Rescue

The Unicode Consortium was started in 1988 with the job of coming up with a single character code set that would be universal (covering all letters, punctuation, and technical symbols used by all world languages), uniform for efficiency, and unique (each bit sequence only has a single meaning). For example, rather than one single

---

1 Extended Binary-Coded Decimal Interchange Code

2 American National Standards Institute

3 American Standard Code for Information Interchange

quote mark there are unique codes for left single quote, right single quote, and apostrophe, because in formal printing they all have distinctly different uses and appearances.

Version 15.1 of the Unicode standard in 2023 defines 149,813 unique Unicode characters, and additional characters are being added as the need arises. Obviously, this requires more than just 8-bits to represent this many unique characters. The basic design is based on 32-bit values, but the possible combinations of bits are cleverly restricted so that all Unicode characters can be represented by one 32-bit value (UTF-32), by one or two 16-bit values (UTF-16), or by one to four 8-bit values (UTF-8). The restrictions on valid bit patterns make it possible to identify whether the next character consists of a single 16-bit or 8-bit value, or if multiple values must be grouped to form one character. A maximum of 1,112,064 valid Unicode code points could potentially be supported, and each code point can be uniquely represented by a 31-bit hex value in the range from 0 to 0x7FFFFFFF. Only a small percentage of those over 2 billion possible hex values map into possible Unicode characters in order to make it easy to distinguish single-value from multiple-value character encodings. The rules for taking a Unicode hex value and producing the corresponding UTF-8 or UTF-16 hex values that would actually be stored are simple for a computer but tedious for a human. Except for the first 128 characters (just like the old ASCII), humans would more likely describe a Unicode character by its hex value rather than the actual byte values that would be stored in a computer memory.

An arbitrary Unicode character can be represented by its hex value using the notation U+xxxx, where "xxxx" is a specific hex value for the symbol. For example, if you do a search for a "Unicode math summation symbol" you will find it is "U+03A3". If you follow your word processor or Operating System rules for entering an arbitrary Unicode character and specify the hex value "03A3", you will get the character "Σ".

The default character code set in use on the most devices prior to Unicode was one of the ANSI code sets, whose first 128 characters are identical to the original 7-bit ASCII standard. Unicode was designed so that the first 128 values of UTF-8 encoding of Unicode are identical to those in 7-bit ASCII and are represented by a single byte in UTF-8. This means that the most commonly used characters are represented by an identical 8-bit value in the ANSI code variants and in UTF-8 encoding of Unicode, so very little space penalty is paid for converting existing English data into UTF-8; and applications designed to support the basic 7-bit ASCII characters could work with UTF-8 by default.

MS Windows started supporting Unicode in Windows NT and in Windows 2000 (as UTF-16 internally). Unicode support for all applications in MS Office was introduced with Office 2000, with partial support as early as Office 97. In 2019 MS began de-emphasizing UTF-16 in favor of UTF-8 for internal API interfaces.

The original FAT[4] file system only supported 6.3 file names in 7-bit ASCII. The VFAT[5] files system extension introduced with Windows 95 supported long file name in Unicode, in addition to a munged 6.3 file name for backwards compatibility. Some characters are still disallowed in file names (\ / . ? * ¥)[6]

---

4 File Allocation Table

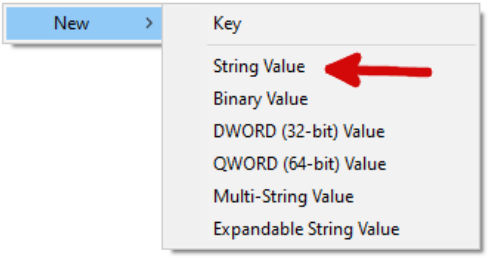
5 Virtual File Allocation Table

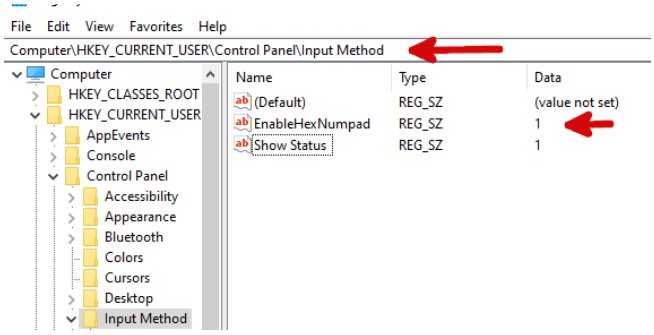
6 The Yen symbol is disallowed because on some Japanese computers the Yen symbol replaced the / symbol in the codeset, causing some ambiguity in how to properly translate that codepoint into Unicode when dealing with legacy data.

Almost all web pages are now encoded using UTF-8. The email protocols and email clients now by default use UTF-8. This greatly simplifies global communication of data including all symbols and markings that are important in other languages. It also means you can now receive SPAM written in Chinese, Korean, Arabic, Russian, etc. and have the characters correctly displayed Sometimes I find it entertaining to feed some of that foreign SPAM to Google Translate just to see what foreign scams look like.

Current versions of Windows and Linux support UTF-8 by default. Some Windows applications implement ways of inserting Unicode characters not on the keyboard by selecting the desired character from a table of more commonly used symbols, or provide a special way to input the hex value for the Unicode character. Both MS Word and Wordpad allow entering a Unicode hex value and then using ALT+x to toggle the hex value into a Unicode character; but that technique isn't available in many other applications, including File Manager, which makes it difficult to use arbitrary Unicode characters for naming folders or files.

There is an alternative Unicode entry method in Windows that doesn't require support in each application, but it does require editing the Windows Registry, which is potentially dangerous if you are prone to finger checks or typos. It requires using the Registry Editor to create a string value under "HKEY_Current_User/Control Panel/Input Method" for EnableHexNumpad" and setting its value to "1", and then rebooting.





I tried this out on Windows 10, and was then able to enter an arbitrary Unicode value in a text field by holding down the ALT key, pressing + on the numeric keypad, typing the hex characters value for the Unicode character, and then finally releasing the ALT key. It appears to work everywhere I tried it, but the need to hold down ALT for a long time is awkward and precludes using touch typing skills. Still, it's the only practical entry technique in some contexts in Windows. I rather prefer the Linux approach, which only requires a brief CTRL+SHIFT+U multi-key combination which is released before keying the hex Unicode value followed by a space.

Even if you are using an application like MS Word which has a fairly large table of special symbols to select, you may find it faster to enter a frequently used Unicode character just by using its hex code rather than using a lot of mouse manipulation to select from a large table of symbols.

If you have reason to believe, or hope, that a special Unicode character exists and can describe it, a Google search combined with "unicode" will probably get you to the corresponding Unicode hex value and an image of the character's appearance. "Unicode solid left arrow" will find all sorts of Unicode arrow characters, including U+2190 ← and U+1F844 🡄. "Unicode pile of poo" will locate U+1F4A9 💩. "Unicode Halloween pumpkin" locates U+1F383, or 🎃. "CJK" is the set of unified ideographs used in modern Chinese, Japanese, Korean, and Vietnamese characters. A search for "CJK unicode home" finds the ideograph U+5BB6 meaning a home: 家 Going to "Google Translate" in the Edge browser and using the ALT+ technique to enter the U+5bb6 Unicode

character in the "Enter text" field, Google recognizes it as "Chinese (Simplified)" with a translation of "Home". You may have to search for a font that supports a specific Unicode character as some lesser used characters are not supported by all fonts.