Unicode Notes

Note: This is a "living document" and is regularly updated. I recommend bookmarking it at one or more of the URLs on page 5 and updating frequently.

Table of Contents

Document Conventions and Styles

There are a number of styles and conventions used in this document which are drawn from numerous sources and now that the document is growing and being shared a little, it is worth mentioning.

Language

This document is written in Australian and British English, not American English. American spelling is only utilised where necessary in a programmatic sense (e.g. setting colour properties to a font in CSS would refer to “color” properties in any examples).

Emacs and Emacs Lisp Terminology

The GNU Emacs convention of referring to various modifier keys has been retained and matched. Those keys being Control, Meta, Hyper, Super, Shift and Escape. When these keys are referred to in Emacs' documentation those key presses are referred to as “C-” for Control, “M-” for Meta, “H-” for Hyper, “s-” for Super, “S-” for Shift and “Esc-” or “ESC-” for Escape.

Obviously some of those don't exist on modern keyboards and supporting them often requires remapping keys, a process made quite feasible by Emacs itself. Details regarding my own reconfiguration of Emacs to enable all of those modifier keys, however, is best left for another document. Particularly when the details vary according to both the hardware and operating system upon which Emacs is being run, it may even vary according to the version of the operating system.

File Format

The file is created with LibreOffice and exported to PDF. Though it is exported with the options to attempt to use the PDF/A-1a format, it does not actually conform to that standard. Multiple features and components within the document are incompatible with the technical specifications of the PDF archive formats.

This is not actually a problem as the sole reason for selecting that export feature is to make sure the relevant glyphs and font data are embedded to render the file, which is very much a requirement given the sheer variety of fonts listed in the document.

Only the PDF version is publicly available for reasons which should become obvious after reading the “Proprietary Fonts and Embedding” subsection of the Font Notes section.

Document Availability

This file is publicly available via Dropbox and Amazon S3 file hosting in multiple regions. The Dropbox location was the original quick and easy download location which are retained as they may have been bookmarked previously. The Amazon S3 URLs are the preferred download locations.

> **Important note:** Amazon S3 servers use a wildcard SSL certificate for all hosted services and will consequently produce SSL or TLS security errors when the hostnames provided do not match. The URLs provided here utilise hostnames based on my domain name of `adversary.org`. In each case the hostname and filepath can be adjusted to utilise the Amazon S3 hostname so as to not see the error message. Alternatively the error may be safely ignored or a HTTP URL may be used.

Organised Adversary Regional Download

Australia

- HTTP: http://files.au.adversary.org/files/UnicodeNotes.pdf
- HTTPS: https://files.au.adversary.org/files/UnicodeNotes.pdf
- AWS S3: `s3://files.au.adversary.org/files/UnicodeNotes.pdf`[1]

Germany

- HTTP: http://files.de.adversary.org/files/UnicodeNotes.pdf
- HTTPS: https://files.de.adversary.org/files/UnicodeNotes.pdf
- AWS S3: `s3://files.de.adversary.org/files/UnicodeNotes.pdf`[2]

United States of America

- HTTP: http://files.east1.us.adversary.org/files/UnicodeNotes.pdf
- HTTPS: https://files.east1.us.adversary.org/files/UnicodeNotes.pdf
- AWS S3: `s3://files.east1.us.adversary.org/files/UnicodeNotes.pdf`[3]

Dropbox

- View online: https://www.dropbox.com/s/8jifzcc8qks5cef/UnicodeNotes.pdf?dl=0
- Download: https://www.dropbox.com/s/8jifzcc8qks5cef/UnicodeNotes.pdf?dl=1

1 https://s3.amazonaws.com/files.au.adversary.org/files/UnicodeNotes.pdf
2 https://s3.amazonaws.com/files.de.adversary.org/files/UnicodeNotes.pdf
3 https://s3.amazonaws.com/files.east1.us.adversary.org/files/UnicodeNotes.pdf

Unicode Version

The current version of the Unicode Standard is 13.0.0; with the specification available via the web,[4] PDF[5] and print-on-demand.[6]

> **Important note:** Major releases of the Unicode Standard are usually annual. The next major release (14.0.0) has been post-poned by six months due to the global Corona Virus pandemic.[7]

Unicode Quick Stats

Current Version:	`13.0.0`	Basic Multilingual Plane (BMP):	`0`
		Supp. Multilingual Plane (SMP):	`1`
Code Points per Plane:	`65,536`	Supp. Ideographic Plane (SIP):	`2`
Assigned Code Points:	`143,859`	Supp. Special Plane:	`14`
Unassigned Code Points:		Private Planes:	`15` and `16`[8]
Special[9] Code Points:		Total Private Planes:	`2`
Practical number of Code Points:	`1,112,064`	Unused Planes:	`11`
Total Code Points:	`1,114,112`[10]	Total Planes:	`17`

Frequently checked references

Tables: `0000`, `0080`, `2000`, `2100`,[11] `2150`, `2200`, `2300`, `2400`, `2600`, `2700`
I Ching symbols: `4DC0`, `2600`; Egyptian hieroglyphs: `13000` to `1343f`;
Emoji: `1f170` to `1f9ff`

Files: A web search for the PDF code charts is available,[12] but a file repository is also available via both the web[13] and FTP.[14] Local copies are kept as an additional reference.[15]

4 http://www.unicode.org/versions/Unicode13.0.0/
5 http://www.unicode.org/versions/Unicode13.0.0/UnicodeStandard-13.0.pdf
6 http://www.lulu.com/spotlight/unicode
7 http://blog.unicode.org/2020/04/unicode-140-delayed-for-6-months.html
8 In addition a subsection of Plane `0`, consisting of `U+e000` to `U+f8ff`, is also for private use.
9 Including control codes and characters, non-characters, UTF-16 surrogates, etc.
10 This includes `0x0`, which is why most software (e.g. `sys.maxunicode` in Python 3) will report this figure as `1,114,111`.
11 In addition to the arrows included in the "Punctuation and Direction" section, the final segments from `U+21b0` to `U+21ff` include many more arrow variations.
12 https://www.unicode.org/charts/
13 https://www.unicode.org/Public/
14 ftp://www.unicode.org/Public/
15 $HOME/Reference/Unicode/UnicodePDFcharts/

Code Point Insertion Methods

Most software inserts Unicode characters by referencing their code point in some way. Usually by the numerical value of that code point, but some software is also able to reference either an old name fore a code point or a current, standardised name.

In order to reduce the number of key strokes code point insertion requires and thus increase efficiency; most software utilises or favours inserting numeric code point values as hexadecimal (base-16) values rather than as normal (base-10) integers.

Hotkeys

LibreOffice macro for entering Unicode:	Shift+Command+U (SHIFT+⌘+U) or F8[16]
Emacs default key sequence for entering Unicode:	C-x 8 RET[17]
Emacs customised key for entering Unicode:	F8[18]
Emacs long form for entering Unicode:	M-x insert-char[19]
Emacs customised key(s) for checking character:	M-F8, F13[20]
Emacs long form for checking character:	M-x describe-char
oXygenXML Editor Unicode hex function:	F8[21]
oXygenXML Editor load file in Emacs:	F9[22]
HexChat IRC client (native OS X):	Shift+Command+U (SHIFT+⌘+U)[23]
HexChat IRC client (X11):	Shift+Control+U
Apple OS X Unicode hex keyboard:	C-SPC (Ctrl+SPC)[24]
Apple OS X UTF-8 plane 0:	Shift+Command+SPACE (SHIFT+⌘+SPC)[25]

16 Default setting for F8 has been moved to F4, which was previously available in LibreOffice (but not available in Emacs as it is reserved with F3 for macro recording).

17 Control+x 8 Return/Enter (C-x 8 RET) followed by either the hex code or the Unicode name.

18 Was previously bound to F7, but changed to F8 in order to match a new setting in LibreOffice preferences.

19 Was previously ucs-insert, which will still work but is deprecated and not recommended. To see information about a character, font and face use M-x describe-char.

20 For several years this was set to F16 with my old Apple USB keyboard, but that keyboard died and has since been replaced by a Satechi keyboard. The Apple keyboard had F1 to F19 keys and I had configured the F13 to F15 keys as cut, copy and paste in multiple applications. The Satechi keyboard, however, only has F1 to F15, but it also has dedicated cut, copy and paste keys. Hence the change in configuration.

21 Hard coded to F8 in a plugin provided to me for version 17.0, a configurable standard feature from version 17.1. I kept it to F8 in order to retain similarity with LibreOffice and GNU Emacs. Unlike LibreOffice and Emacs the oXygenXML method requires entering the four characters and then pressing the key or key sequence, whereas with LibreOffice and Emacs the key (F8) or key sequence is pressed and then the hexadecimal characters are entered. Only Emacs supports entering more or less than four hexadecimal characters, but it's also the only program capable of multiple font fallback configuration to cover the full range of UTF-8 characters.

22 Customised External Tool option in oXygenXML Editor which loads the active file in a new Emacs instance (i.e. not emacsclient connecting to a running Emacs server or daemon).

23 Unfortunately it was not possible to configure it to use or alias F8 to this function and match the other interfaces.

24 https://apple.stackexchange.com/a/275053 — Use M-<3-byte characters>; use the Meta / Alt / Option key with a four character, hexadecimal code point entry.

Apple OS X UTF-16 all planes:	C-SPC ; M-<hexadecimal>[26]
GNU/Linux Unicode insertion:	Shift+Control+U
Free/Net/OpenBSD Unicode insertion:	Shift+Control+U[27]

The generic OS X unicode insertion methods[28] leave a great deal to be desired, especially by comparison to the options available by default on BSD, GNU/Linux, and other UNIX systems. On the other hand, the platform independent GNU Emacs provides, without a shadow of a doubt, the best and most efficient means for entering unicode characters into anything.

25 https://apple.stackexchange.com/a/293124 — Enter the 3-byte hexadecimal code directly in the character search table which pops up.

26 https://apple.stackexchange.com/a/183056 — Use M-<4-byte+ hexadecimal>; use the Meta / Alt / Option key with a four character, hexadecimal code point entry.

27 Effectively this is, and always has been, the default key sequence on UNIX for entering at least everything in plane 0 and, depending on the systems, may also enable entering plane 1 characters (including emojis).

28 https://apple.stackexchange.com/q/183045 — The initial StackExchange question which may have revealed the only feasible means of achieving this across the board within OS X.

Unicode and File or Data Formats

Unicode and XML

Issues with XML and UTF-8 are well documented within the XML specifications and which are mainly utilised to address the use of greater than ("<") and less than (">") symbols in writing element tags.

It should be noted that XML also supports UTF-16, it's just that UTF-8 is usually the most common. Both UTF-8 and UTF-16 cover the full range of 1,112,064 code points. Most of the rest of this document deals with UTF-8 only and, with a few exceptions (e.g. Emoji), the first three bytes.

A number of these issues are addressed in different ways in different types of XML usage. For physical and electronic printing, where a word processing format is not used, as with OpenDocument Text (ODT) or Office Open XML (DOCX), I am most likely to utilise DITA and specifically the DITA for Publishers (D4P) specialisation.[29]

It may be worth checking the W3C document, *Unicode in XML and other Markup Languages*.[30] Though that document has currently been withdrawn as a Unicode Technical Report by the Unicode Consortium, it was originally a joint publication by both the Unicode and W3 Consortiums. It does contain some useful information in one location, particularly regarding reserved code points, but many of the references or citations are out of date so it should only be used as a pointer or reminder when checking something.

Unicode and the Darwin Information Typing Architecture (DITA)

As DITA is a form of XML, just refer to the main reference material regarding Unicode and XML with either UTF-8 or UTF-16.

Unicode and DITA for Publishers (D4P)

This is the same as with DITA, refer to the Unicode and XML material.

Unicode, HTML, XHTML and XML

XHTML is always treated the same as XML because it is XML; including support for both UTF-8 and UTF-16. HTML's conformance with XML varies depending on the version, but generally HTML versions prior to 4.01 will cause problems with both Unicode and XML, whereas above that HTML 4.01 support matches XML.

See below for version specific details.

29 I've already begun contributing code to and bug-fixing for the latter. Mainly with the HTML 5 and EPUB plug-ins.
30 https://www.w3.org/TR/unicode-xml/

Unicode and HTML 3.2 and earlier

This precedes the creation of XML and did not inherently rely on either UTF-8 or UTF-16. Character encoding formats varied between sites and specific extended Latin character sets were often specified using built-in code point references, usually as integers rather than hexadecimal.

Unicode and HTML 4.x

This coincides with (or immediately precedes) the creation of XML and subsequently XHTML 1.0 and 1.1. It also coincides with the introduction of styles and CSS.

Unicode and XHTML

This is the same as with DITA and any other XML dialect, regardless of whether it is XML 1.0 or 1.1. Most XML files use XML 1.0 anyway.

Unicode and HTML 5

This adopts the use of UTF-8 as the standard character encoding format from XML. Also extends the use of CSS to CSS 3.0.[31]

Unicode and HTML 5 with XML

Sometimes referred to informally as XHTML 5 or XHTML5. It's basically everything that comes with HTML 5, except utilising a strict XML format as was the case with XHTML. This makes it easier to use it with XSLT transformation rules to convert into other XML data or convert other XML formats into it.

Unicode, HTML, XML and Ebooks

There are multiple ebook formats, some open standards and some not. Most of those in use, however, are built on various types of markup; including XML, XHTML and an assortment of HTML versions.

PDF and PostScript are not included here since their principal purpose is to produce printed documents, even though they can be read electronically.

31 Really that version number will be dropped and CSS components will be versioned independently of each other, making all CSS significantly more modular.

Unicode, HTML 3.2 and Open E-Book Publication Structure (OEBPS)

This shouldn't be seen anymore, but unfortunately due to the existence of Amazon's Mobipocket format, this is still haunting us (see below). This is because the Mobipocket format is essentially the ancient Open Ebook format encased in DRM.

OEBPS 1.0

The Open Ebook format (OEBPS 1.0) was originally just a single HTML 3.2 file with a little bit of CSS 1.0 shoe-horned over the top (and embedded in style element tags), plus any images are base64 encoded and embedded in that HTML. It is possible that it may have incorporated HTML 4.0 or a subset of HTML 4.0 with early style information.

Version 1.0 could not have included any XML or XHTML because it pre-dated the creation of both of those standards. Published in September, 1999.

OEBPS 1.0.1

Published in June, 2001.

OEBPS 1.2

Published in August, 2002.

OEBPS 2.0

This was basically EPUB 2.0 (see below).

OEBPS and Mobipocket

The Mobipocket format just takes the initial release of OEBPS 1.0, saves it encoded in the old Microsoft character encoding format of Windows-1252 (see below), adds DRM encoding and changes the extension of the filename.

Due to the use of the Windows-1252 character encoding, Mobipocket is incapable of providing any Unicode support. This means all extended characters are limited to those characters supported by both that old Microsoft character encoding format and those characters directly supported by the underlying HTML standard of the OEBPS version utilised by Mobipocket.

Ever wondered why Mobipocket files have so many limitations? Including the reason for the inability to embed a font or fonts and why there are restrictions on what languages Amazon permits publishing in (i.e. English language books only)? This is why.

The only "feature" Mobipocket provides which neither OEBPS nor EPUB provide is DRM built-in by default.[32] This provides nothing of value to consumers, readers and authors; while being of

32 Adobe's DRM added to EPUB actually breaks the specifications, although EPUB 3.0 and above do provide provisions for obfuscation of embedded fonts, without applying DRM to the content.

questionable or limited value to the publisher (i.e. Amazon).[33]

Unicode, XHTML and EPUB 2.0.1

EPUB 2.0 and 2.0.1 is essentially built on XHTML 1.0, 1.1, HTML 4.0 and CSS 2.0 or CSS 2.1. Additional CSS features may be supported depending on what else the device or ereader software supports.

This is also where the EPUB name was introduced and the reason for it being version 2.0 and above is because version 1.0 was the old Open Ebook format.

Unicode, HTML 5, XML and EPUB 3.0.1

For example, this is the format used with EPUB 3.0.1 files and the strict definition of HTML with XML enables greater standardisation of generating standards conforming EPUB output from another XML format via XSLT conversion methods. Indeed this is precisely how DITA for Publishers (see above) generates EPUB files from the same source XML material which can also produce various other file formats (including, but not limited to, HTML, XHTML, ODF, DOCX, Adobe InCopy XML format,[34] PDF via FOP,[35] PDF via XHTML+CSS renderers,[36] HTML WebHelp with JavaScript search and many more).

Unicode, HTML 5 and EPUB 3.1

EPUB 3.1 was published in 2017 and mostly deals with shifting certain "experimental" features to use specific parts of the current CSS standards instead of ad-hoc implementations required in previous EPUB versions.

Unicode, HTML 5 and above EPUB 3.1

It is possible, even likely, that future versions of the EPUB standard will not require strict XML compliant markup and will follow the HTML standard. It is, however, quite likely that it will still accept formats which retain the strict XML markup.

In these cases it may be possible to have a compliant EPUB 3.1 or higher format which does not include support for UTF-16. Whereas EPUB 3.0 and 3.0.1 will support any form of Unicode that the component technologies support and since that includes XML then it includes either UTF-8 or UTF-16.

33 There are no technical merits and effectively no value added in regards to the business relationships between either Amazon and consumers/readers or Amazon and authors. Their adherance to this technological relic is solely aimed at maintaining a "walled garden" environment which artificially isolates Kindle users from open standards.
34 Apparently it's easier to generate InCopy than InDesign files directly.
35 For example: Apache FOP (free), XEP/RenderX and Antenna House (as expensive as it comes).
36 For example: Prince XML (reasonable-ish price, produced locally in Melbourne) and Antenna House (as expensive as it comes).

Embedding Fonts in EPUB Files

Though there is limited support for font embedding in EPUB 2.0 and 2.0.1, this was significantly improved with EPUB 3.0 and above. Indeed, that is one of the many reasons to utilise the more recent EPUB standards. Some aspects of this may still be supported with an EPUB 2.0.1 file, but that will depend more on the hardware or software used to display the file than anything else.

The EPUB standard only permits embedding fonts in the Open Type Font (OTF) format and the Web Open Font File (WOFF) format. Though some software and ereaders may still display the True Type Font (TTF) format, it is technically not part of the standard and may result in validation errors. It is recommended that any TTF files which are required by an EPUB be converted to OTF or WOFF first.

This is why I have converted the Symbola font from TTF to OTF. This converted file is available from my site under the same terms as the original release (i.e. freely available for any use).[37]

Font Obfuscation

This is a contentious topic in some circles as it is viewed as similar in some respects to DRM, which is not permitted in the EPUB standard, mainly due to a similarity in implementation. The key difference between the standardised font obfuscation and Adobe's DRM for EPUB files is that the technical details are publicly available. It can certainly be reversed, though it is somewhat fiddly to do and largely not worth the effort.

Font obfuscation is a means by which a publisher may embed a proprietary font in such a way that they would not be in breach of the licensing terms for that font. This would enable, for instance, utilising the same fonts as might be used in a corresponding print edition of the same book.

Though it is possible to do this, it does create a number of additional problems which may make it not worth the effort. Those problems include the fact that correct display of an obfuscated font is not supported by all ereader devices and software, so even if it is done correctly it might not work for most readers. The Kobo devices, for instance, do not display obfuscated fonts, even though their software is based on Readium, which can display an obfuscated font.

If using an obfuscated font at all, my recommendation is to also include a font which is permissively licensed enough to embed without obfuscation to fallback to for those devices or software which can't use the obfuscated font. This is done in CSS in the same way as any font selection is done.

Embedding Fonts for Apple iBooks

Apple's standard ebook format is a slightly modified version of EPUB and is utilised in iBooks for both OS X and iOS. Supporting the quirks of iBooks and the Apple variations is quite possible without breaking the EPUB standard. For instance, including a smaller version of the cover image

37 OTF conversion available here: http://files.au.adversary.org/files/fonts/Symbola.otf

for display in the iBooks library screen is simply a matter of including the reference in the OPF file with a specific element type.

For iBooks to utilise an embedded font, whether obfuscated or not, it is simply a matter of including an XML file in the `META-INF/` directory visible in the root of the EPUB file. This is the same directory which contains an XML file pointing to the location of the OPF file. The OPF file will contain the details of the embedded fonts themselves, just as it contains their locations and other relevant details. If the publishing software does not support adding this file when the EPUB is created, it is fairly simple to add manually. Since this XML file is added to the `META-INF/` directory, it does not need to be included in the manifest found in the OPF file.

The XML file must be named `com.apple.ibooks.display-options.xml` and it must contain the following XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<display_options>
  <platform name="*">
    <option name="specified-fonts">true</option>
  </platform>
</display_options>
```

Once added it will have the location `META-INF/com.apple.ibooks.display-options.xml` in the EPUB file.

Recommended Fonts for Embedding

The best font or fonts to embed in an EPUB will largely depend on the type of book it is and what the embedded fonts are intended to do. In general the main reasons for embedding fonts are to either provide coverage for glyphs which are not available in common fonts, like Times New Roman or Georgia, or to meet aesthetic preferences of the author or publisher.

For instance in fiction my preference for print publishing is to use Adobe Systems' Garamond Premier Pro, but the license for that font would not permit embedding without obfuscating the font files. Due to the problems which arise with font obfuscation as described above, I find it is generally better to exclude that font family from the corresponding EPUB file.[38] In these cases I use and recommend the Cormorant Garamond font, which is available through Google Fonts under the SIL Open Font license.

From time to time I've also had the need to include coverage of more obscure glyphs and in those cases the font or fonts selected for the task will vary according to the support for the glyph in question. For an *I Ching* hexagram (䷖) I would usually use Symbola, for the Bitcoin currency symbol (₿) the only option is Noto Sans Symbols and other glyphs may have other requirements.[39]

38 Not counting any draft versions produced solely for my own use during the editing process and which are never distributed.

39 Astute readers will notice the slight increase in line spacing for this last line of the paragraph, caused by the mixing of Noto Sans Symbols with Times New Roman; even though the Noto Sans Symbols characters have been reduced to 10pt while the Times New Roman text remains at 12pt.

Unicode and LaTeX

LaTeX[40] is the weird, but strangely effective cousin of modern publishing, unfortunately it's also affected by some rather serious and significant drawbacks; some of which are related to Unicode and some aren't. Furthermore while it has a highly technical and dedicated userbase, the majority of whom believe it to be the best typographic solution in all cases, these same people are generally the ones who refuse to meet the publishing needs of those outside of their own particular niche.

The inevitable consequence of this attitude and technical development path is that for the specific niches of that cadre of loyal users, principally scientists, engineers, mathematicians and computer science students, it excels. Particularly for books and documents requiring the presentation of mathematical or chemical formulæ and engineering diagrams or schematics. For years LaTeX was really the only viable option for these things and anyone using anything else ultimately had to resort to inserting images of such things in their publications to display them properly.

To do this sort of thing, however, TeX, LaTeX and its assorted other variants, including LuaTeX and XeLaTeX (AKA XeTeX), made design decisions which took it in a very different direction to the rest of the publishing industry. Not that there's anything wrong with that in and of itself, particularly not when TeX's foundation was laid before certain aspects of current standard formats had been invented. Yet it is very much the case that these design decisions and approach had a long and far-reaching effect.

One of the most significant of those factors being the development of its own font format designed specifically for the needs of it's technical userbase and the particular niche they occupy within the wider publishing community. Another of those significant and certainly fundamental factors is that the TeX file format is also its own programming and printing language, a language with the sole purpose of publishing information and from its initial design to do so in printed format.

It's not a bad approach, indeed the current publishing industry standard for doing so is a format designed entierely to make a predictable, on screen file format which would automatically be generated by the printing language which first became the *de facto* standard and which is now an ISO standard. That file format being the Portable Document Format (PDF) and the printing language which generates it being PostScript. The problems with LaTeX, however, stem from the fact that there are vast sections of its function and development which is at odds and in many cases even oppositional to the approach taken by Adobe during the early development of PDF, PostScript and its predecessors.

The rise of electronic book formats, particularly the EPUB standards, are additional aspects which TeX and LaTeX do not cope well with. From a basis in printing documents only and doing so in a manner at odds with the rest of the world, it is does not handle conversion to SGML, XML or HTML well and as a consequence it does not handle the ebook formats based on either HTML or XML (or both) well either.

Then, of course, there's the matter of the fonts. The issue is not that LaTeX has its own font format,

40 Pronounced "lay-tek" and this document will be instantly recognisable to LaTeX users as not being published using it by the typesetting on the term LaTeX there. It's likely that this section alone will demonstrate why not, but I'm sure someone will insist it can meet all my needs without grasping what those needs are. To that individual, whoever you are: remake this entire cheat sheet in LaTeX alone, including all the embedding and the complete TOC. Then, if you achieve that minor miracle, make a second copy that adheres to the PDF/X-3:2002 standard (and no cheating by outputting to PostScript and then opening that in a pre-configured Adobe Distiller).

nor is it that that font format is almost entirely unusable outside of LaTeX. No, the issue there is the strong resistance within LaTeX to adopting either TrueType fonts or OpenType fonts for a long time. It has only been relatively recently, perhaps the last ten years or so, that that has become at all feasible and even now it is something which a lot of people attempting to use LaTeX struggle with.

It has also hampered the ability to support UTF-8 content within LaTeX documents and publications. Regardless of whether or not it was the intent of those immersed in the LaTeX world, it certainly seemed to those outside it and looking in that the LaTeX view was that anything not already LaTeX isn't worth publishing and of no value. Indeed, it seemed very much that this recent capitulation only came after the near universal acceptance of Unicode globally and the subsequent adoption of Unicode, particularly UTF-8 as the standard right down to the filesystems on the computers on which LaTeX is installed.

So now they're playing catch up and discovering that when a system is developed in total opposition to something else which subsequently becomes a standard, then the first system is left with numerous historical relics which increasingly just seem to be deliberate hurdles. Not being able to use the fonts needed to display or print the characters in the Unicode standard, is a pretty big hurdle these days. Even now the default LaTeX engine for producing PDFs still can't handle OpenType fonts. Only LuaLaTeX, itself arguably closer to the Lua language than it is to LaTeX, and XeLaTeX are capable of processing OpenType fonts. Only XeLaTeX is able to do so with any degree of ease.

Surely that's good enough, though, right?

Well, if you're a university student more concerned about handing in essays than in the real publishing world beyond your local campus, then maybe so. If, however, you need to deal with the professional publishing and printing world then you know there's a few other standards to pay attention to, most importantly in the form of the PDFs which ultimately go to industrial printers. There are also enough LaTeX books, mainly textbooks and technical documentation, which make it into print to know that somewhere in the different types of TeX processing there is at least one component which understands those standards and is reasonably well documented enough for it to be in use on more than one continent.

Here's the thing, though, that component is pdfLaTeX and not either LuaLaTeX or XeLaTeX. So you can either have a system which produces the correct file format to go to print or you can have a system which handles the fonts with the full character coverage to meet the needs of as many languages as possible on this little planet, but you can't have both.

Faced with this sort of situation and a great deal more besides it, those who did not have the specific requirements of mathematicians and engineers simply chose to skip the frustration and anguish. Some went to Adobe, others to QuarkXPress (until it imploded and they went to Adobe too), others went to various XML platforms when that became available, initially using Docbook. Some even developed their own custom solutions; but the one constant was that without a specific requirement necessitating it, no one outside of that academic engineering and technical niche chose LaTeX if they actually had a choice.

The proponents of LaTeX may periodically make various claims regarding it's superior typographic and typesetting technology and implementation. Maybe, when deployed by someone who genuinely knows it inside out that is even true.[41] The fact is that any past advances were not worth the price of converting to LaTeX and the more time which passes, the greater the improvements amongst the non-LaTeX publishing world such that if there was a technical lead in LaTeX then that gap is narrowing until it is simply not relevant a point of comparison. The relevance of LaTeX itself will diminish and likely fade into obscurity sometime thereafter.

Aside from the character encoding and communications issues which centre on Unicode itself, as well as other publishing standards and requirements outside of the engineering and technical professions (e.g. publishing fiction, arguably a moderately notable segment of the market), there's another recent development which is the proof of the narrowing of that gap.

For decades TeX has been the principally accepted way to properly typeset algorithms and formulæ, but now it's not the only kid in the schoolyard. Now that MathML has reached a point of maturity to move beyond just some XML implementations to part of the specifications for HTML 5 along with gaining support in standard office applications, including LibreOffice and Microsoft Office, that monopoly has been rather thoroughly broken.

The LaTeX crowd would no doubt then cite the fine grained control available in creating diagrams or any other geometric shape in the context of typesetting. To which the markup languages, led once again by XML, respond with SVG.

The days of LaTeX providing genuinely unique typographic features to a publication are gone, but the complexity, difficulty and frustration with it remain. Unsurprisingly people will go with what works, what they can make work for them and, in a professional context, what they can get support for. In the majority of cases LaTeX won't even get a look in and where it does the majority of that will avoid dealing directly with LaTeX or will only use it as one step in a chain of other automated systems. The majority of which will generally be produced via Pandoc conversion filters.

There probably is still some niche use for it for a while yet. I certainly still deal with it in the context of certain specific projects, but that use generally does not extend to general purpose document production and it certainly doesn't extend to professional publishing or printing projects. All of that generally feeds into an XML based system or systems.

One thing the TeX systems does have in its corner is a large repository of software brought together from disparate projects and toolkits. For this reason, if nothing else, there are still valid reasons for installing the TeX software suites (e.g. tex-live), even if the TeX format is never utilised.

> **Important note:** A full installation of the TeX software suite will generally include a copy of the Ghostscript PDF and PostScript software. As of late 2013 the free software version of Ghostscript and MuPDF switched from the GNU General Public License to the GNU Affero General Public License.

If you have any concerns with the use of this license in that software potentially affecting your own projects, it may be necessary to consider using one of the last releases of Ghostscript which was released under the GNU GPL. That would be Ghostscript 9.09 or 9.10 and both are available from the old SourceForge download site.[42]

41 Given how few of those documents even adjust the page margins to something reasonable, such experts are not in the majority.
42 https://sourceforge.net/projects/ghostscript/files/GPL%20Ghostscript/

PDF and PostScript

Adobe's Portable Document Format (PDF; .PDF, .pdf) and PostScript (PS; .PS, .ps) are the backbone of modern printing and publishing. There's a lot of issues with them, architectural relics from their design and even hardware limitations from the 1980s or so.

Nevertheless, they do manage to produce what they most essentially need to do. Furthermore, while some more recent file formats may be technical improvements, the indisputable fact is that PDF and PostScript have become industry standards and if one's intention is to publish and not worry so much about the technical limitations of the industry, then one is better off just working with it when necessary.

PostScript

This is both a file format and a programming language. It is literally the set of instructions which tells a printer what to print and how to do so, as well as defining things like the page size and so on.

This is why PostScript files must be converted to Extended PostScript (EPS) or PDF in order to be viewed.

PDF/A-1

Also known as ISO 19005-1:2005; PDF/A-1 is designed for long-term archival purposes. Like PDF/X-3 (see below), this format requires embedding any font data needed to render or print the file.

This includes both PDF/A-1a and PDF/A-1b.

PDF/X-3:2002

Also known as ISO 15930-3:2002; PDF/X-3:2002 is the industry standard for graphic and textual printing press quality publications. Images are rendered at 300 DPI or higher, usually between 300 and 450 DPI. The format includes those components of the fonts used that are necessary to print the whole document.

This is the reason for font licensing embedding exceptions in for printing documents. It has less to do with the transfer of printing data from a computer to a peripheral or networked printer and more to do with meeting the requirements of the publishing industry.

> **Note:** The PDF/X-3:2003 format is a superset format containing every feature of the PDF/X-3:2002 format and while it can be used during the course of print ready document production; the final file for delivery is generally best explicitly reprocessed to the standard PDF/X-3:2002 format.

File Names and Locations

For the most part the degree of support for Unicode characters in file names and other components of locating or referencing a file is determined by what the relevant operating system or systems can support, some other hardware or software directly involved in accessing the file and some other software or protocol specifications may be a factor.

In English speaking IT systems there remains a tendency to adhere to naming conventions which retain ASCII characters only in the naming conventions largely to reduce the potential problems which might arise if the file were to be accessed from a system significantly more limited than is commonly the case with modern operating systems.

Additionally there are a number of ata types utilised within core protocols and data specifications which explicitly define the uniform resource identifier and location types (i.e. URIs and URLs, respectively) and when they must always be able to be represented entirely as an ASCII text string. These limitations were originally purely technical, but though it is now possible to overcome many of the initial constraints from computing during the mid twentieth century, there are now political, policy related and other practical aspects of retaining aspects of the initial limitations.

The practical aspects include providing a minimum specification for standardised access regardless of origin, hardware or other factors. There is a certain kind of irony in the fact that very western and pro-English data standards on which these protocols were built, which guarantees a desgree of standardisation and simplicity of processes and systems which will help ensure as near as possible the broadest range of technical access to modern globally interconnected networks.[43] Even while the users of those systems continue to cite the necessity of the opposite in the sophistication of the applications and specifications they interact with directly when using those systems.

The Domain Name System and Unicode

One of the most obvious points of contention online with the technical issues of ASCII specifications and human requirements for non-ASCII information being available is in the domain name system. Particularly when those domains and their websites are often the most prominent reference points by which people associate or identify particular issues, businesses, organisations and even people.

It is understandable, then, that those non-English speakers, often also referred to as the vast majority of the population of Earth, would want to access these things in their own native languages. Even while the domain name system remains one of the fundamental core systems with corresponding protocols and additional systems built on it which require it essentially remain ASCII only.

The question then became, how could the underlying technological system upon which so much else relied be maintained, while simultaneously providing the non-English scripts and non-Latin glyphs in domain names? The answer or possible answer was what is now known as punycode, from RFC 3942, the title of the RFC defining it, *Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*.[44]

43 To reference the the full term from which the proper noun, "the Internet," was derived just once.
44 https://tools.ietf.org/html/rfc3492.html

Essentially it is simply a means of converting a Unicode code point with a value of 128 or above in alphanumeric ASCII text. In addition to RFC 3942, there is additional detail regarding the protocols affected by it in RFCs 5890[45] and 5891.[46]

Since the DNS specifications already prevented two hyphens (or hyphen-minus) appearing together in a domain name or permitting a domain name either beginning or ending with one; the hyphen or hyphen-minus was selected to provide the means of indicating a hostname or fully qualified domain name (FQDN) or part of one contained punycode.

Each part of a FQDN, that is the alphanumeric section between the dots in the FQDN, is a segment which either permits or does not permit the use of punycode according to the policies set for that namespace by the relevant authoritative source for it. Though this does mean that each domain name license holder can effectively set policy and implement a broader range of Unicode than might otherwise be permitted within their domain name's parent namespace's official policies, this will have less adverse effects in that context as there will still be a final authoritive point of contact in each domain.

How this will be represented or displayed in each browser is likely to vary considerably. Some may or may not even be able to handle either domains containing Unicode characters or the punycode string.

Most languages have some means of running through the byte conversion process available to them, most of which are simply implementing the sample code included in one of the earliest RFCs on the topic. My current preferred means of resolving these domains comes by way of the GNU IDN Library; both libidn and libidn2.[47] In spite of the the existence of a number of relevant code points in the multilingual planes above 65,535; neither libidn nor any of the other implementations I've experimented with process punycode or IDN strings with a value beyond the first plane of UTF-8.

Even so, there's still a great deal of potential in both the multilingual glyphs and other symbols in various parts of that first plane. Especially in segments like the ever popular `0x2600` to `0x26ff` range. Indeed my first successful test of the use of a punycode subdomain within a standard ASCII domain name within the .org gTLD namespace was a character from that range of symbols. It was one of the planetary symbols.

Most browsers, including text only or text mainly use the punycode to access the URL, but display the URL using the preferred unicode glyphs. The exceptions appear to be Chrome and chrome based browsers (e.g. Opera), which only display the punycode URL, Firefox or other Mozilla and/or Gecko based browsers, which require further testing. Safari for OS X, w3m[48] and eww[49] all display the intended URL once the site loads. Testing has been minimal and limited thus far and the tests have solely dealt with only the most basic DNS configuration with an A record set and pointing back to the main website of the parent domain.

45 https://tools.ietf.org/html/rfc5890
46 https://tools.ietf.org/html/rfc5891
47 https://www.gnu.org/software/libidn/
48 A text based browser frequently used with Mutt and other console applications.
49 A text based browser implemented with elisp for use within Emacs.

Unicode and Programming Languages

Unicode and Python

In Python 2.x using escape characters with various encoding functions (e.g. appending `.encode("UTF-8")` after a string containing the escape characters is required). Python 2.x only supports the first three bytes of UTF-8, as can be seen by checking the value of sys.maxunicode (`65,535` or `0xffff`).

> **Note:** Python 2.7 reached its EOL on January 1st, 2020. Python 2 relevant information is only retained in case working on legacy Python systems is needed.

In Python 3.x there is better support for Unicode. Strings only require the use of a different type of escape character which utilises the 4 character hex code as listed below. For example, a string containing `\u00a9` will produce the copyright symbol. This will always work with Python 3, but in most cases it is unnecessary. This method only works for code points up of up to three bytes, that is from `0x0000` to `0xffff`. For code points above plane 0, use the extended format with a capital "U" and the 8 character extended hexadecimal format (e.g. `\U0001f926`).[50]

Python 3 includes the ability to enter characters using their official name. It also has the ability to return the corresponding integer of a code point, in either hexadecimal or glyph format, with the built-in `ord` function.

```
>>> "\u263f"
'☿'
>>> "\N{MERCURY}"
'☿'
>>> ord('☿')
9791
>>>
```

This can then be used to create more emoji-like characters with variation selectors and other character modifiers (e.g. `"1\N{variation selector-16}\N{combining enclosing keycap}"`).

As long as the encoding method is specified in the file, it is possible to enter the UTF-8 character directly. Checking sys.maxunicode in Python 3 reveals the maximum value for UTF-8 code points (`1,114,111` or `0x10ffff`). See the Python documentation for more details.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

Including the following starting lines in Python 3 scripts does no harm. Python 3 ignores "from __future__" imports because it knows that the purpose is to backport Python 3 functions to Python 2.7, though effectiveness in 2.7 may depend on the micro release version. Some things might even be backported to 2.6, but there are no guarantees.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

50 This extended 8 character hexadecimal format supports plane 0 as well, it's just less efficient to also use it for that.

```
            from __future__ import unicode_literals
            from __future__ import print_function
```

Some of Python 3's Unicode support has been backported to Python 2.7.[51] Use the unicode_literals attribute or method in the `__future__` module to extend that.

Encoding and decoding the unicode characters in a manner similar to that utilised by Python 2 remains possible, but is no longer needed to be utilised constantly. Unicode characters in a string will display as intended, but encoding that will produce an extended, byte-encoded string with all the unicode characters beyond ASCII displayed as extended hexadecimal code. Simply decoding that byte-encoded object returns the characters to their expected appearance.

```
>>> a = "æ"
>>> b = a.encode()
>>> b
b'\xc3\xa6'
>>> c = b.decode()
>>>
```

In addition to these built-in functions, the `unicodedata` module[52] provides more detailed reference and character checking functions. The support for the Unicode standard in this module in Python 3's standard library is significantly greater than Python 2 ever had. For example, Python 3.8.2's version of the module supports Unicode 12.1.0,[53] whereas Python 2.7.17 only supports Unicode 5.2.0.

The two most useful methods in the `unicodedata` module is `unicodedata.name` and `unicodedata.lookup`. Though looking up hexadecimal UTF-8 code points and the unicode glyphs will produce a KeyError and using a name check on the Unicode character or glyph name will produce a TypeError. See the module documentation for further details.

```
>>> unicodedata.name('\u263f')
'MERCURY'
>>> unicodedata.name('☿')
'MERCURY'
>>> unicodedata.lookup('MERCURY')
'☿'
>>>
```

The `foad.py` script[54] provides plenty of informative examples with UTF-8 characters. Third party modules, like the emoji module[55] and the grapheme module,[56] can also improve useability. Storing code point references in JSON files[57] can also be leveraged easily.

51 Some has even been backported to Python 2.6, but there is absolutely no excuse or reason for using that anymore. Anything using Python 2.6 or earlier should have at least been ported to Python 2.7 or have the option for doing so available. Since this is even true of the *"Unofficial Patch" for Vampire: the Masquerade – Bloodlines*, which shipped with Python 2.1 and has an optional mod to update it to use Python 2.7.2 then nothing else has any excuses.
52 https://docs.python.org/3.8/library/unicodedata.html
53 Unicode 13.0.0 was released after Python 3.8's initial launch and addition of the 13.0.0 specifications might not be included in a point release.
54 https://github.com/adversary-org/foad (NSFW).
55 https://github.com/carpedm20/emoji
56 https://github.com/alvinlindstam/grapheme
57 https://gist.github.com/Vexs/629488c4bb4126ad2a9909309ed6bd71

Greater detail is available in the highly recommended Python *Unicode HOWTO*.[58]

Python and Punycode

A commonly utilised Python module for quickly converting domain names containing support for non-ASCII Unicode characters is the `punycodeurl` module available from PyPI via pip. This module depends on the `encodings.idna` module in the standard library, which can also be called directly. The `encodings.idna` module is also automatically loaded with a number of other core components of the standard library, including socket and asyncio.

Unfortunately `punycodeurl` is only available in the soon to be EOL'd Python 2.7.[59] So it's a good thing that I've already ported it to Python 3. Usage is identical to the original and it is currently available via a branch of my forked repository[60] or via the pull request.[61]

Alternatively there are the third-party `idna`[62] and `idna_ssl`[63] modules available via PyPI. The `idna` module is available for Python 2.6, 2.7, 3.3 and above. While the `idna_ssl` module is only available for Python 3.3 and above.

Nevertheless, there is greater or more consistent support for international or multilingual domain names through more comprehensive and specific libraries, such as GNU's IDN Library.[64]

Python Handling of URIs, URLs and URNs

Conversion of various types of characters to and from hexadecimal URI encoding is achieved via parser in the `urllib` standard module.[65]

```
>>> import urllib.parse
>>>
>>> url = "https://www.dropbox.com/s/8jifzcc8qks5cef/UnicodeNotes.pdf?dl=1"
>>> uri = urllib.parse.quote(url, safe=":/")
>>> uri
'https://www.dropbox.com/s/8jifzcc8qks5cef/UnicodeNotes.pdf%3Fdl%3D1'
>>> urllib.parse.unquote(uri) == url:
True
>>>
```

58 https://docs.python.org/dev/howto/unicode.html
59 **Note:** This limitation is solely in regards to a very simple and easy means of calling the standard library's encodings.idna module functions; it is not meant to imply that Python 3 has no punycode support, nor does it mean that I'm reimplementing the whole damn thing in Python. That said, if I really had to then I'd just use one of the methods of providing access to C libraries, such as CFFI, with the GNU IDN Library.
60 https://github.com/Hasimir/punycodeurl/tree/2to3
61 https://github.com/charsyam/punycodeurl/pull/1
62 https://github.com/kjd/idna
https://pypi.python.org/pypi/idna
63 https://github.com/aio-libs/idna-ssl
https://pypi.python.org/pypi/idna_ssl
64 Like the GNU Privacy Guard and a number of other essential GNU projects, libidn is dual licensed under both the GPL and the more permissive LGPL.
65 Only references to Pythin 3.x are included here in anticipation of Python 2.7 being designated as EOL in 2020.

Lisp and Emacs Lisp

Those aspects of elisp which behave more in the nature of editor functions and are bound to two of the function keys, are best left to the section on Emacs itself below.

While it is possible to display minimal information about a character at any given code point, specifically either its name or old name if it has one, using Emacs Lisp (elisp); I generally don't see much value in that. More often than not I need the extended data available from the `describe-char` function; particularly the code point values themselves and the font face data, including the font size in use.

Nevertheless, this is the elisp to display a character's name:

```
(get-char-code-property ?⛧ 'name)
```

This is the elisp to display an old name which pre-dated the character's inclusion in the Unicode standard:

```
(get-char-code-property ?⛧ 'old-name)
```

Selecting the entire function and running the `eval-print-last-sexp` (bound to `C-j`) will print either the name or the old name of the character.[66]

Indeed, the elisp behind `insert-char` (`F8`) is this:

```
(insert-char CHARACTER &optional COUNT INHERIT)
```

While the elisp for `describe-char` (`M-F8` or `F4`) is this:

```
(describe-char POS &optional BUFFER)
```

Both of which are able to take insertion or cursor points interactively to perform their functions, unlike the preceding minimal functions.

Additionally the GNU libidn C libraries provide elisp minor modes containing functions for handling IDNs and punycode from within Emacs. Those elisp files are copied to the relevant site-lisp path during compilation and installation. Alternatively the idn command can be called via a shell, either within Emacs or not.[67]

66 Additional discussion of this can be found on StackExchange: https://emacs.stackexchange.com/questions/4121/how-can-i-programmatically-get-the-canonical-unicode-name-of-a-character

67 Run `idn -h` for the details, but generally `idn -a $char` and `idn -u $punycode` will do what most people want.

Scheme and Guile

Scheme is a modern and very popular dialect of Lisp with Guile being the GNU Project implementation of it. As it is part of the GNU Project, there are particular advantages for using it with Emacs as well.

The majority of Scheme source files[68] utilise either ASCII text or Unicode text. Due to the requirement for native support for ASCII, however, this means that Unicode support may only be provided using UTF-8 and **not** UTF-16 or UTF-32. By default, if no character encoding is explicitly specified, UTF-8 will be used.

Since explicit is always better than implicit, this method of declaring the character encoding is encouraged:

```
#coding:utf-8
```

While Guile is case insensitive for this type of declaration, Emacs expects such declarations to be lower-case. So consistently using that, even when not using Emacs is highly advisable.

To enter a specific character[69] in hexadecimal format in order to match the other examples in this file, the method is to enter `\x0` to `\x10ffff`. Though it is also possible to use octal numbers and character names (in either a Scheme or Lisp naming convention or the Unicode Consortium's official names of any given character).

The easiest way to make any character appear, however, is simply to enter the character directly via a relevant editor or IDE, like this:[70]

```
(display "☥")
```

This is the advantage which comes with setting the character set to UTF-8 or, more to the point, not changing to a character set which does not support Unicode.

Greater detail is available in the *Character Encoding of Source Files* section[71] of the *Guile Reference Manual*.[72] Along with the sections on strings,[73] strings as bytes,[74] character sets[75] and the standard character sets.[76] No doubt some other data types and their handling in Guile may be of use as well.[77]

68 File extension: `.scm`.
69 https://www.gnu.org/software/guile/manual/html_node/Characters.html
70 This time using the ANKH as the test subject instead of the INVERTED PENTAGRAM.
71 Section 6.18.8 of the current release documentation (Guile version 2.2.3).
72 https://www.gnu.org/software/guile/manual/html_node/Character-Encoding-of-Source-Files.html
73 https://www.gnu.org/software/guile/manual/html_node/Strings.html
74 https://www.gnu.org/software/guile/manual/html_node/Representing-Strings-as-Bytes.html
75 https://www.gnu.org/software/guile/manual/html_node/Character-Sets.html
76 https://www.gnu.org/software/guile/manual/html_node/Standard-Character-Sets.html
77 https://www.gnu.org/software/guile/manual/html_node/Data-Types.html

Unicode and Operating Systems

Unicode and Mac OS X

Enabling the Unicode Hex Input in System Preferences / Keyboard / Input Sources allows entering Unicode as hex code in any application or terminal. To activate the alternative input source or toggle between it an the primary input source (usually the country or regional keyboard or US keyboard), use the Command+Shift+Space or Command+Space key sequences (or select it in the Apple menu bar). Once the Unicode Hex Input source is active, any plane 0 hex sequence can be entered while holding down the Alt/Option key.

In general, this should be a fallback option for when standard key bindings or preferences do not meet the requirements. Mostly this should be for entering Unicode characters on the command line or some similar command line program where Emacs is not being used (e.g. when using my command line Twython Tools[78]).

Additionally most special characters and unicode characters can be accessed within OS X applications from the bottom of the Edit menu. In older versions of OS X this menu option is called Special Characters, but in more recent versions of OS X it is called Emoji & Symbols. The screen can be accessed using the Command+Ctrl+Space key sequence.

OS X also supports a custom, platform specific subset of characters which can be used with certain key sequences using Alt/Option and other modifiers with it. This is similar to the Windows specific key sequences in that operating system, except the characters entered that way remain Unicode characters (UTF-8) and do not use an alternative character encoding. The penultimate section of this file shows all the alternative characters which can be entered in this manner.

Unicode and Linux

Most if not all Linux distributions are able to input UTF-8 characters anywhere through the Ctrl+Shift+U key sequence in a manner similar to OS X uses with the Command key.

Unicode and BSD

Expect it to be the same as for Linux for the most part. It may have some similarity to OS X, but that is less likely this time as this is one of the aspects where Apple has changed things (usually due to Cocoa).

Unicode and Windows

Like Apple, Microsoft use their own codes in conjunction with the Alt key, apparently also providing a UTF-8 system-wide input method. I care very little about this.

78 https://github.com/adversary-org/twython-tools

Microsoft is also responsible for many multilingual character set conflicts which became common during the 1990s. The most notorious of these being Windows-1252, which is still in use in the Mobipocket (.MOBI) file format on Amazon Kindle devices. This is the leading cause of content conversion conflicts to Amazon's ebooks. The problems caused by these character sets are solved with Unicode and the use of either UTF-8 or UTF-16.

Unicode and Software

Unicode and Emacs

As with all things related to Emacs, it is eminently customisable. Which I have already set to some extent with the binding of `F8` (and no longer using the Apple functions as the default in favour of the actual F-functions being primary). The default method cited in most guides is `C-x 8 RET`, but I find that far too protracted given how often I use Unicode characters.

Emacs supports smart font selection where it will use the first located font which supports any given glyph in order to display the correct glyph. Since that selection depends on using the first font it finds which claims to support a particular code point and since it searches them in what appears to be alphabetical order,[79] it is still possible to have some characters not display properly. Fortunately it also enables specifying fallback fonts for particular code point ranges.[80]

Displaying all of the installed and recognised fonts in Emacs is possible by running the following elisp code in a buffer, like the scratch buffer using the `C-x C-e` command:

```
(dolist (font (x-list-fonts "*"))
  (insert font "\n"))
```

Additional support for improved handling of Unicode characters can be provided by Roland Walker's packages: unicode-fonts,[81] ucs-utils,[82] font-utils[83] and persistent-soft.[84] The packages are also available in the MELPA repository, but that repository has a number of problems which may not be appropriate for many users. Not least of which being the management policy of the repository, which has been known to deliberately break or sabotage end user configurations as part of internal disputes with package developers. As a consequence I do not recommend using MELPA and am in the protracted process of moving away from it as much as I am able.

Currently set to use `Inconsolata` followed by `Source Code Pro`, with multiple fonts used for fallbacks with different ranges of glyphs. This includes a great deal of coverage of the multiplanes from code point 65,536 (`0x10000`) and above.

Entering Unicode characters by hex code with a value above 65,535 (`0xffff`) is supported. This covers most Emojis. Relevant key bindings include `insert-char` to `F8` and `describe-char` to `F16` and `M-F8`.

A copy of relevant parts of my `$HOME/.emacs` file is appended to the end of this file.

79 The documentation actually says its font selection method is somewhat random.
80 https://stackoverflow.com/a/6084198/2801707
81 https://github.com/rolandwalker/unicode-fonts
82 https://github.com/rolandwalker/ucs-utils
83 https://github.com/rolandwalker/font-utils
84 https://github.com/rolandwalker/persistent-soft

Unicode and Atom

This is a terrible choice and favoured only by those who have drunk the Node.js Kool-Aid. Still, it may be necessary from time to time to enter that catastrophically bad ecosystem. Native support for entering hexadecimal code points is not great and doing so requires a third party package. This is true of most Atom functions. By default the package which provides this function is configured to use the standard Command + Shift + U or the Control + Shift + U. It is, at least fairly simple to add more key bindings and thus I was easily able to add F8 to performing that task. I also replicated my "dedicated" cut, copy and paste keys as F13, F14 and F15, respectively.

Unfortunately it manages multiple font fallback configurations about as effectively as Hexchat and is consequently fairly useless. I avoid it as much as possible and only note it here for when it can't be avoided. Even Vi (as distinct from Vim), Pico and Nano are better choices than Atom.

Unicode and Vi and/or Vim

You're in the wrong place for this. The purpose of Vi is to perform a system recovery when booting off a floppy disk since it will fit on a 1.4MB floppy. By the time you actually need to deal with non-ASCII characters, the system ought to be able to boot a full operating system. At this point you can use a real text editor, like Emacs.

As for Vim, I will just repeat the shit-stirring line I pulled out at a consultancy many years ago when certain wet-behind-the-ears Comp. Sci students were waxing lyrical about Vim:

"Vim? Vi Improved? Isn't that Emacs …"

Unicode and LibreOffice

This file serves as a very good example of just what is possible and how since it was written in LibreOffice and then exported to PDF.

UTF-8 characters may be entered manually by pressing `F8` and then entering the four character hexadecimal code point. This is made possible by the old Compose Special Characters extension,[85] which definitely still works with LibreOffice 5.0.x, 5.3.x,[86] 5.4.x and 6.0.x.

Entering Unicode characters by hex code with a value above 65,535 (`0xffff`) is not supported. Characters with values above 65,535 require locating the character in a special characters chart.

85 http://www.productivityapps.com/csc.html
https://extensions.openoffice.org/project/ComposeSpecialCharacters
https://extensions.openoffice.org/en/download/4646

86 Versions 5.1 and 5.2 were skipped because they broke many essential user preferences.

It is, however, possible[87] to enter hexadecimal code points of two or three characters if those characters are solely followed with spaces which make up for the missing characters. So entering the Greek letter psi could be done by pressing the code point insertion command and following it with `03a8`.

With this method, however, one presses the code point insertion key(s), enters `3a8` and then the `SPC` key. The resulting character would still be "Ψ" and the space would need to be pressed a second time to insert an actual space character.

Performing the same task with the copyright symbol is very similar. The standard method would be pressing the insertion key(s), followed by `00a9`. With the alternative method one enters the insertion key(s), followed by `a9`, followed by two `SPC` key presses to produce "©" and then follow that with any actual spaces.[88]

Unicode and oXygenXML Editor

Support for UTF-8 encoding matches that of LibreOffice and Emacs; under the hood it apparently utilises UTF-16, but it does have trouble with some fonts.

Entering Unicode characters by hex code with a value above 65,535 (`0xffff`) is not supported. Characters with values above 65,535 require locating the character in a special characters chart. It is also possible to copy and paste the character in from another program or to load the file in an external editor, such as Emacs (see above) to enter the character.

The character will only be displayed if the font in use supports that character or if a CSS file explicitly calls a font which supports the character (e.g. Symbola or Noto Sans Symbols). Noto Sans Symbols and Symbola are known to work.

Code2000 does not display properly at all, only half of each glyph is rendered and the same thing occurs with Code2001 and Code2002. The Apple font for rendering Emojis causes the underlying JVM to crash. It has not yet been tested with some Google fonts, including Noto Emojis. Nor has it been tested with the Twitter Color Emojis font (see below).

See also the section(s) on XML, DITA, D4P, XHTML and HTML above.

Unicode and IRC

IRC Unicode support is generally determined more by the font type support and font selection methods of individual clients rather than the protocol itself. My IRC use is mostly with HexChat and ERC in Emacs, but occasionally I load either an old version of XChat or one of the other Emacs clients. Usually my connections are managed through a single ZNC instance, but it appears to have

87 At least as of LibreOffice 6.0. I am unsure of earlier versions.
88 It should surprise no one that this very recent feature discovery was the result of a typo while working on another document.

no trouble passing Unicode characters back and between the IRC servers and my IRC clients.

Unicode and HexChat or XChat

Does not provide smart glyph selection, instead relies on manual configuration of primary font and alternative fonts.

Use of the alternative fonts configured by XQuartz HexChat (X11) is inconsistent with loading some characters correctly. As a consequence the default primary font has currently been changed from 16 point Roboto to 15 point Noto Sans Symbols. The latter is just as clear and easy to read as Roboto, but with far greater glyph support, though not quite as nice as Roboto.

XQuartz version of hexchat installed via MacPorts retains original special key inputs, so it uses the Control key instead of the Command key. Aqua or Cocoa builds of either HexChat or XChat use the native Apple keys (i.e. the Command key instead of the Control key).

Manually entering Unicode hexadecimal code points is possible with either the Control+Shift+U key sequence or the Command+Shift+U key sequence, depending on IRC client version.

Entering Unicode characters by hex code with a value above 65,535 (`0xffff`) works, but is still dependent on glyph support in the configured font(s). Support for Emoji characters is limited.

Unicode Utilities

The most useful Unicode specific utilities I've found thus far; not counting programming languages, features of other programs (e.g. Emacs) or anything reliant on an external site or service.

Unum

This is simply brilliant. It appears to be nearly complete up to the last Unicode 13.0 release, is well maintained and is freely available for any use.[89] There are two versions, a compressed one[90] and an uncompressed one.[91] I opted for the latter and since even uncompressed it's still only 8MB, that's insignificant.

It's just a Perl script which contains the data from the entire Unicode Standard wit which it converts between numeric code points in octal, decimal, hecadecimal, binary as well as XHTML or HTML escaped characters, Unicode code point name, the Unicode character itself, the block in which the character is assigned or by any regular expression which matches any of those.

Matches will display output of the code point or code points in octal, decimal, hexadecimal, HTML,

89 https://www.fourmilab.ch/webtools/unum/
90 https://www.fourmilab.ch/webtools/unum/download/unum_comp.tar.gz (737KB download, 960KB install size).
91 https://www.fourmilab.ch/webtools/unum/download/unum.tar.gz (1.4MB download, 8MB install size)

the character itself and the Unicode name. It is, however, a command line program only and it will still be subject to the available fonts where it is run. It also has a few quirks as a result of its versatility, but they're easy enough to get used to. Particularly since it is well documented via both its home page and its perldoc man-style page. The latter is obtained with this command:

```
bash-4.4$ perldoc unum.pl
```

Using the PENTAGRAM as the example as used previously demonstrates this:[92]

```
bash-4.4$ unum.pl n=pentagram
   Octal  Decimal      Hex        HTML    Character   Unicode
  023344     9956   0x26E4     &#9956;    "⛤"         PENTAGRAM
  023345     9957   0x26E5     &#9957;    "⛥"    RIGHT-HANDED INTERLACED PENTAGRAM
  023346     9958   0x26E6     &#9958;    "⛦"    LEFT-HANDED INTERLACED PENTAGRAM
  023347     9959   0x26E7     &#9959;    "⛧"         INVERTED PENTAGRAM
bash-4.4$
```

As some might guess, leaving the preceding `n=` instruction out will result in output for each letter in the word or phrase used unless the search term matches one of the numeric values.

Running it in a shell within Emacs is quite effective as it benefits from the highly customisable and complete or as complete as possible font support of that editor. Unlike the built-in features of Emacs, however, it is more readily scriptable for bulk conversion of code point values to characters.

Unicode and Related Subject Matter Experts

Joel Spolsky on Developers and Unicode

This isn't so much regarding specific software as it is essential reading for any developer of any software which must deal with character encoding at some point. Which effectively means all developers and all software. This is, in fact, required reading for any developer having trouble with UTF-8 or even trying to avoid accepting or switching to using UTF-8 and/or UTF-16 at all.

It's just one little article, but it is one of the most succinct and useful for any struggling developer:

> *The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)* by Joel Spolsky; 8th of October, 2003.[93]

Dan Sugalski on C, Unicode, strings and character encoding

These are more closely tied to specific languages, primarily lower level ones, like C, but deals with a number of the same sort of problems which Joel Spolsky wrote about. There is no indication as to whether or not Sugalski was aware of Spolski's article, even though only three days passed between Spolski's and the first of Sugulski's being published.

92 Output adjusted slightly for better readability.
93 https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/

Also note that the third of these three articles has *far* less to do with unicode and character encoding than the first two, but anyone already finding those first two articles directly relevant to their work will likely find the same is true of the third.

1. *What the heck is: A string* by Dan Sugalski; 11th of October, 2003.[94]
2. *Strings, some practical advice* by Dan Sugalski; 14th of October, 2003.[95]
3. *What the heck is: A type* by Dan Sugalski; 29th of November, 2003.[96]

The USA's National Security Agency (NSA) on PDFs and Metadata

The NSA hasn't really got the best reputation on this little spinning rock of ours. Certainly not with most people, which is largely due to the fact that the vast majority of the world's population are not American and thus are within the scope of the NSA's charter to spy on (and so they do). That said, there's no arguing the fact that they are genuinely experts in matters of file metadata and file structures and analysis.

So it should come as no surprise that when I needed to delve deep into the innards of the PDF format, it ended up being a small NSA file I stumbled across that was of the most benefit. They subsequently removed it from their website, but that doesn't prevent me from redistributing it; particularly when I've already cited it in a previous report.[97]

> *Hidden Data and Metadata in Adobe PDF Files: Publication Risks and Countermeasures* by the Enterprise Applications Division of the Systems and Network Analysis Center (SNAC), Information Assurance Directorate; 27th of July, 2008.[98]

The file is also available from the same hosts and directories as this file, except for Dropbox. So that's the USA,[99] Germany[100] and Australia.[101]

> **Note:** The usefulness of the NSA document is likely to be limited for most readers, except perhaps out of some mild curiosity regarding what the NSA might glean from a PDF beyond the words written in it. Particularly for those whose principal concern is in preparing documents for either print or electronic distribution.

94 http://www.sidhe.org/~dan/blog/archives/000255.html
95 http://www.sidhe.org/~dan/blog/archives/000256.html
96 http://www.sidhe.org/~dan/blog/archives/000276.html
97 Like all US Government documents and resources which have been voluntarily released at some point, as opposed to material which has been leaked instead of declassified, this NSA document is in the public domain and not subject to any copyright claims at all.
98 http://okbounty.adversary.org/pdf_risks.pdf
99 https://files.east1.us.adversary.org/files/pdf_risks.pdf
http://files.east1.us.adversary.org/files/pdf_risks.pdf
s3://files.east1.us.adversary.org/files/pdf_risks.pdf
100 https://files.de.adversary.org/files/pdf_risks.pdf
http://files.de.adversary.org/files/pdf_risks.pdf
s3://files.de.adversary.org/files/pdf_risks.pdf
101 https://files.au.adversary.org/files/pdf_risks.pdf
http://files.au.adversary.org/files/pdf_risks.pdf
s3://files.au.adversary.org/files/pdf_risks.pdf

For those who may find it necessary to dig deeper into the arcana that is the PDF specifications, however, this document is invaluable. Regardless of whether or not the motivation for doing so is to solve some obscure print production error or to perform a data forensics analysis on a file or files. The latter scenario being what led me to discovering this little gem in the first place.

Conversions and Transformations

One of the most inevitable aspects of dealing with all of these things is the need to turn one type of document into another type and the options for this are many and varied. What's more, they're constantly expanding.

Emacs Modes

Htmlize

The Htmlize[102] mode attempts to transform an Emacs buffer or file into HTML 4.01, including replacing UTF-8 characters with their corresponding integer based code point. The colour properties of the Emacs session itself will be preserved in CSS inserted within the generated HTML, but is commented out by default.

> **Note:** This Emacs mode may be the key to future transformation of properly conforming EPUB 3.0.1 files to the dumbed down content types necessary to produce Kindle Mobipocket files (aka the US editions).

Org Mode

Emacs' Org Mode[103] is a text based format which is considerably more feature rich than Markdown, but perhaps not quite as feature rich as reStructuredText. The built-in export and publishing options provide options for generating fully validating XHTML or converting to LaTeX, amongst other things. It also leverages LaTeX to produce PDFs and can include additional headers to pass to the selected LaTeX engine in order to streamline that PDF production.

A good example of the use of Org Mode to produce both XHTML web documentation and PDF documentation for download is the GnuPG projects documentation and website. The entire website is written in Org Mode, as are the documents in each software package in the project.[104]

The following headers at the top of a file will generate a 6" by 9" book with the Garamond Premier Pro font at 11pt and a half-inch margin:

```
#+LATEX_COMPILER: xelatex
#+LATEX_CLASS: book
#+LATEX_CLASS_OPTIONS: [11pt]
#+LATEX_HEADER: \usepackage{xltxtra}
#+LATEX_HEADER: \usepackage[paperheight=9in, paperwidth=6in, margin=0.5in]{geometry}
#+LATEX_HEADER: \setmainfont[Ligatures={Common}]{Garamond Premier Pro}
```

Wheras these headers will produce an A4 size article with the Times New Roman font at 12pt with a one inch margin:

102 https://github.com/hniksic/emacs-htmlize
103 https://orgmode.org/
104 https://gnupg.org/

```
#+LATEX_COMPILER: xelatex
#+LATEX_CLASS: article
#+LATEX_CLASS_OPTIONS: [12pt]
#+LATEX_HEADER: \usepackage{xltxtra}
#+LATEX_HEADER: \usepackage[a4paper, margin=1in]{geometry}
#+LATEX_HEADER: \setmainfont[Ligatures={Common}]{Times New Roman}
```

This uses LaTeX's default font in conjunction with ligatures, a 12 point font and 1 inch margin, while retaining the regionally appropriate paper size:[105]

```
#+LATEX_COMPILER: xelatex
#+LATEX_CLASS: article
#+LATEX_CLASS_OPTIONS: [12pt]
#+LATEX_HEADER: \usepackage{xltxtra}
#+LATEX_HEADER: \usepackage{fontspec}
#+LATEX_HEADER: \usepackage[utf8]{inputenc}
#+LATEX_HEADER: \usepackage[margin=1in]{geometry}
#+LATEX_HEADER: \setmainfont[Ligatures={Common}]{Latin Modern Roman}
```

In spite of my criticisms of LaTeX, which still stand, I clearly found some of those solutions. Since LaTeX is used to generate PDFs for other projects I am involved with, this aspect of using Org Mode is a very useful feature to me.

Additionally, Org Mode is capable of exporting or converting to Texinfo and Emacs Info formats, as well as to man pages. Utilising the Texinfo export option then lends it to the additional output format generation available from the Texinfo software directly (see below).

Pandoc

Pandoc[106] is a favourite of anyone who writes documentation in largely text based formats like Markdown, reStructuredText and Org Mode. Even if it's not used to prepare final file formats, though some use it that way, it's an essential tool to have available.

Since oXygenXML Editor now includes a feature to convert from Markdown to DITA, Pandoc is now able to feed into that system by converting other text formats, especially reStructuredText and Org Mode. The former being the go to documentation format for Python projects and the latter being the go to documentation format for Emacs and Elisp projects.

Docutils and reStructuredText

105 In spite of the name and description of `inputenc` with utf-8, either in conjunction with the `fontspec` package or not, it does not actually provide coverage of the entire Unicode standard in the same way as XML or as HTML and CSS can since it does not provide the same facility for font fallback support. The only way to address such issues is by manually editing the LaTeX output to insert specific font overrides for every single glyph missing from the main font selected for a document. This is fine for those who are more comfortably using LaTeX than anything else and who are not generating large amounts of documentation in an automated way or providing documentation which would be processed into LaTeX post-distribution, but otherwise it is a serious problem.

106 http://pandoc.org/

Docutils[107] and reStructuredText are the mainstays of documentation within the Python language development. They are the building blocks on which the Sphinx documentation generation server is built and which powers both the official Python documentation[108] and the Read the Docs website.[109]

Docutils also includes its own generic XML format which it generates from reStructuredText text and which converts seamlessly back to reStructuredText.

> **Note:** While Sphinx's output for online documentation is usually very good, using it as a publishing platform with its EPUB option is not viable for any kind of professional work. The EPUBs generated frequently result in hundreds of validation errors and the last time I checked they were still only generating EPUB 2.0.1 files.

Texinfo

Though heavily related to LaTeX; Texinfo[110] is its own separate thing which has been the backbone of the GNU Project documentation efforts for many years. It is designed to take its own custom source format (`.texi` files) and build end user documentation from that in multiple formats.

The most common output formats Texinfo produces are: HTML, Emacs Info, manual pages, PDF and Docbook 4.2. It is also able to build or convert to its own, custom XML format and that conversion process is bidirectional so the Texinfo XML format can be converted back to the source format. Though that format is little less frequently used, it ought to be able to be leveraged by any other XML format to transform to and from Texinfo via XSLT files.

XML and XSLT

This is arguably the most important in my production processes. XML and XSLT 2.0 is arguably the basis on which major XML formats transform the XML source to other XML formats, including XHTML, to HTML and other formats. DITA and DITA for Publishers leverage it extensively and thus I'm essentially calling on these processes every time I generate an EPUB file or initiate the processes which ultimately result in producing a print ready PDF.

PrinceXML

PrinceXML is produced by Yes Logic in Melbourne Australia.[111] Prince takes a different approach to many in rendering PDFs from HTML or XHTML and CSS. It also provides a unique adaptation of ligature and hyphenation preprocessing adapted from LaTeX, but without the other problems which come with utilising LaTeX directly. It's very easy to use, it also understands the difference between rendering for electronic output with RGB colours and print output with CMYK colours.

107 http://docutils.sourceforge.net/
108 https://docs.python.org/
109 https://readthedocs.org/
110 https://www.gnu.org/software/texinfo/
111 https://www.princexml.com/

While there are plenty of specific sample types available on the PrinceXML sample page,[112] one of the best means of showing off its capabilities is in *The Magic of Prince* example. In particular by examining the XHTML file[113] with its embedded CSS and the corresponding PDF file[114] produced by Prince with that XHTML file.

Prince also renders PDF output for multiple PDF ISO standards, including the PDF/X-3:2003 format; a more advanced version of the press ready standard used in publishing. Documents generated by Prince are often very close to what would be needed for a final document, but not quite. It's proprietary, but is free to use for non-profit purposes with a watermark. A commercial license is a few hundred dollars.

XSL-FO and FOP

A variation on XSLT, a Formatting Objects Processor utilises XSL Formatting Objects in conjunction with XSLT and Xpath to transform XML documents to another format, usually PDF. Continued development of HTML and CSS is ultimately intended to replace this entirely, but since HTML and CSS do not yet meet all the output types that XSL-FO is capable, there is still use for it in some use cases.

Antenna House

This is the Rolls Royce of XML and HTML rendering and PDF processing. Antenna House have multiple options and products, but essentially two approaches.[115]

One branch of their products performs stylesheet based processing and rendering with the stylesheet formats being either XSLT or CSS.[116] The other branch of products utilises their own, custom XSL-FO[117] and FOP transformation methods.[118] Additionally they're actively involved in DITA development, mainly because so many of their customers use DITA to prepare the content before it is processed by Antenna House's products.[119]

The formatter for both branches or approaches is probably their leading software and the commercial only licensing is extremely expensive.[120] They are, however the primary reason why so many of the documents produced at the enterprise level around the globe are of the quality they are. They also specialise in multilingual document transformations, particularly the languages of east and south-east asia (they're based in Japan).

112 https://www.princexml.com/samples/
113 https://www.princexml.com/howcome/2016/samples/magic6/magic.xhtml
114 https://www.princexml.com/howcome/2016/samples/magic6/magic.pdf
115 https://www.antennahouse.com/antenna1/
116 https://www.antennahouse.com/antenna1/css/
117 https://www.antennahouse.com/antenna1/xml-to-xsl-fo-stylesheets/
118 https://www.antennahouse.com/antenna1/xsl-specification/
119 https://www.antennahouse.com/antenna1/dita-editor/
120 https://www.antennahouse.com/antenna1/formatter/

Apache FOP

This is the only completely free FOP software package available, but updates are infrequent and it is usually not capable of the sort of things done with commercial FOP products. Especially Antenna House's products.

SyncRO Soft Chemistry

From the authors of oXygenXML Editor, this provides basic PDF generation using either XML or HTML 5 input with CSS and driven by Apache FOP. It is currently in beta and freely available.[121]

RenderX/XEP

RenderX's XEP engine provides an XSL-FO/FOP processor to generate PDF and some other output formats.[122] Like PrinceXML, a free, watermarked version is available except their watermarked output is more intrusive. Using it is a bit more complicated than Prince's solution and a commercial license is more expensive. I tried it a few times, but found the experience to be quite unsatisfactory, especially given the price.

HTML Tidy

HTML Tidy[123] isn't so much a file conversion tool as it is a cleaning tool. It processes HTML, XHTML and XML files to automate the process of making them compliant with whichever of those formats is specified. Basically it removes the validation errors which may be introduced during other conversion processes.

PDF to PostScript

This provides a means of doing to PDFs what HTML Tidy does for HTML, XHTML and XML in a sense. For a variety of reasons, particularly in automated processes, it may be that the final, or near final PDF has managed to acquire some problems. Usually of a nature that doesn't show up in its appearance, but does prevent passing the preflight checks of a given PDF type.

The quickest and easiest way to deal with a problem like this is to convert the file to PostScript and then convert it back to the target PDF format through some other preconfigured or scripted method. Using Adobe Distiller for this purpose is probably the easiest, but it can be done with GhostScript and some other tools, depending on which PDF target format is intended.

When it comes to command line conversion tools, there's basically two options: `pdf2ps` and `pdftops`.

121 https://www.oxygenxml.com/chemistry.html
122 http://www.renderx.com/tools/xep.html
123 http://www.html-tidy.org/

The first of these, pdf2ps, is basically just a shell script wrapper for GhostScript; it is somewhat inconsistent in the quality of the PostScript files it produces and thus the PDFs produced from those PostScript files also suffer from quality issues. In general those files will still pass preflight checks, but may be significantly larger than they might otherwise need to be.

The second of these, pdftops, is part of the Poppler PDF rendering library.[124] In general it produces higher quality PostScript output than the GhostScript wrapper script. It also supports additional modifications at runtime in a way that the GhostScript method does not. Since files would most likely already have the correct output size set by this point, running it with the following options would be wise:

```
pdftops -level3 -origpagesizes -noshrink -nocrop $PDF_FILE $PS_FILE
```

Not Production Ready

Anything in this section is not fit for use in any professional publication, but they're frequently well known and there will always be questions about them.

Amazon Kindlegen, CreateSpace and KDP

The Amazon supplied or provided conversions are a special case; being simultaneously not fit for producing professional quality material and the only ways to produce ebooks of a format which may be sold on the Amazon websites.

On top of which, the CreateSpace channel has recently been proven to be used as a means of facilitating fraud and money laundering. Computer generated "books" are listed at massively inflated prices and subsequently purchased using stolen credit card data by the fraudsters. Thus enabling them to receive the 60%-70% royalties Amazon pays to (American) authors.[125] This type of activity under the Amazon banner certainly casts a shadow over those authors only using the Amazon services and may also do so to those whose work is available through other channels in addition to Amazon's CreateSpace.

> **Note:** Amazon announced in August, 2018 that CreateSpace would be closed within "a few weeks" and that all CreateSpace customers would be migrated to KDP. It is unclear as to what that final date will be, but it is probably intended to be completed before the Thanksgiving holiday in the United States.

The newer KDP brand is aimed at pursuing more of the traditional publishing market and while there are plenty of scam related criticisms; the majority of those appear to be the usual efforts by Amazon to rip off authors rather than enabling the kind of exploitative gimmicks made possible by CreateSpace.

The easiest solution in such cases is to only sell hard copy books via Amazon, optionally offer EPUBs everywhere except Amazon and ignore Kindle entirely. Depending on the genre or the

124 http://poppler.freedesktop.org
125 https://krebsonsecurity.com/2018/02/money-laundering-via-author-impersonation-on-amazon/

target market, it is quite possible that pandering to the demands of Jeff Bozos[126] and his commercial empire is a waste of time and effort anyway. This is particularly the case for authors of non-English material as well as for material which has been translated from English to any other language.

Calibre

This ebook management software is a must have for consumers of ebooks in whatever format they are obtained in. The goal of Calibre is solely to make all formats accessible to their owners and the focus is end user access to content and ***not*** to provide a production ready platform for ebook, and particularly EPUB, preparation.[127]

Though many people recommend Calibre for exactly that purpose, especially amongst or to self-publishing authors; my view is that it should never be used to produce EPUBs or any other format for publication. I do, however, use it to test EPUBs I produce and, of course, to manage my personal EPUB library. It's also the only software I use to manage my ereaders, not least of which being to keep all my ebooks from multiple sources managed in one location.

Sigil

Sigil used to have a brilliant reputation and it was brilliant before the EPUB 3.0 specification was released.[128] It provided the means to edit and validate EPUB 2.0.1 files, as well as provide an easy way to enter useful metadata in the OPF file.

Then the EPUB 3 specification came along and Sigil's decline began. First with a protracted lack of support for the new standard and then, even when development did resume it managed to get that implementation so badly wrong that opening an EPUB 3.0 or 3.0.1 file in Sigil and then saving that file again would *always* ruin the file.

It was sometime during this period that I first toyed with writing my own toolkit, but ultimately[129] learned of and subsequently dove right into adopting DITA and DITA for Publishers to produce EPUBs instead. The abject collapse of Sigil to being in any way functional and useful with regards to EPUBs is also what resulted in me turning to trying and then purchasing oXygenXML Editor.[130]

wkHTMLtoPDF

A Google project utilising WebKit to generate PDF and image files from HTML and CSS.[131]

While this project does work, particularly with the image files generated, it runs into trouble when

126 Apparently "Bozos" is not his name, but it's possible that important data became misrepresented because of his insistence on not utilising a common, open standard for communication (e.g. Unicode).
127 https://calibre-ebook.com/
128 https://sigil-ebook.com/
129 Following a rant on IRC on the topic.
130 This was the first time in about two decades that I had been so absolutely turned off by the abject failure of a FOSS project to embrace a proprietary one. Since then I've discovered just how far ahead of Sigil (and pretty much everything else) oXygenXML really is for all kinds of XML, HTML and other markup languages.
131 http://wkhtmltopdf.org/

creating PDFs with certain types of fonts that it has trouble rendering properly. One might consider it a cheap attempt at PrinceXML, but to do so does Yes Logic a grave disservice.

This is good enough for playing around to produce quick and dirty hacks, as I have used it for in the past on Twitter,[132] but no more than that.[133]

132 http://www.adversary.org/wp/2015/10/03/so-you-want-to-tweet-longer/
133 Before the 140 character limit was raised to 280 characters.

Font Notes

Note that some characters in the charts here needed to be manually set to Code2000 or Noto Sans Symbols in order to display correctly, where previously the OS X automatic font selection worked.[134] This appears to be due to LibreOffice overriding the OS X font selection defaults with the Asian Text font instead of continuing to fail over to the next font with an appropriate glyph.

This also means that it may be necessary in some projects to embed fonts like Code2000 in order to make things display correctly. Such circumstances are likely to be few and far between, but if they do occur it will be necessary to check the relevant font licensing under such circumstances.

The only font to provide glyphs for all of the first plane of the Unicode specification is Unifont, but since it is computer generated it looks very blocky and ugly. Most people find the font too unattractive to even consider using, but it is the ultimate fallback if there is no other choice.

This is an example of Unifont.

It's pretty clear to see the lack of aesthetic appeal there. There is a related and slightly smoother alternative and that is:

The Unifont Upper font.

There should only be very rare times to use the main Unifont typeface and probably no reason to use the Unifont Upper variant. It should also be noted that there are no italics or underlined versions of either of those, with the Upper variant acting as the bold typeface.

Code2000, Code2001, Code2002, Symbola, Droid Sans Fallback, Noto Sans Symbols, Apple Symbols and Arial Unicode MS should be checked *well* before resorting to Unifont. Droid Sans Fallback has slightly less, Arial a similar amount, Noto Sans Symbols has more recent Unicode Standard additions and the Code2000 series have many more code points.

Proprietary Fonts and Embedding

Most proprietary fonts include exceptions to embedding restrictions to permit printing. Meaning that embedding or transmission of the font is permitted to produce a printed document which may then be distributed, but does not permit embedding of the whole font in a way which would also act as distribution. So proprietary fonts could be used in the production of a document saved in either the PDF/X series, including the publishing standard of PDF/X-3:2002, or PDF/A series formats;[135] which embed only those parts of a font needed to render and print the document, but could not be used in the production of an EPUB without DRM or font obfuscation.

134 Note that the automatic font selection might not have worked really, but the fallback font by default is usually Arial Unicode MS and it covers a great deal of the Unicode specification. It may simply have been that glyphs which are not present in Arial Unicode MS were not amongst those I initially tested for around the time this document and its predecessor were written.

135 It would also apply to PostScript (.ps) or Extended PostScript (.eps) formats with embedded font data. The same applies to SVG files where the font used has been converted to SVG font, character or glyph objects.

Fonts ranked by Number of Glyphs

This table lists the fonts with the top number of glyphs as well as the basics of the license to which they are subject.

Font	No. of Glyphs	License	Origin
Code2000[136]	63,546	Shareware	James Kass
Arial Unicode MS	50,377	Proprietary[137]	Microsoft Corporation
Droid Sans Fallback	49,775	Apache 2.0	Google
C o d e 2 0 0 2[138][139]	30,469	Shareware	James Kass
Adobe Fangsong Std[140]	30,284	Proprietary[141]	Adobe Systems Inc.
Adobe Heiti Std[142]	30,284	Proprietary[143]	Adobe Systems Inc.
Adobe Kaiti Std[144]	30,284	Proprietary[145]	Adobe Systems Inc.
Adobe Song Std[146]	30,284	Proprietary[147]	Adobe Systems Inc.
Adobe Fan Heiti Std[148]	19,156	Proprietary[149]	Adobe Systems Inc.
Apple SD Gothic Neo	18,662	Proprietary[150]	Apple Inc.
Adobe Gothic Std[151]	18,352	Proprietary[152]	Adobe Systems Inc.
Adobe Myungjo Std[153]	18,352	Proprietary[154]	Adobe Systems Inc.

136 http://files.au.adversary.org/files/fonts/CODE2000.ZIP
137 Personal use, no embedding or distribution. Permission for printing.
138 Unrelated to the first two and adds a significant portion of the remainder the Unicode specification.
139 http://files.au.adversary.org/files/fonts/CODE2002.ZIP
140 No **bold** or *italics*, mostly Chinese characters.
141 No listed terms, but visible copyright.
142 Entirely in **bold** and no *italics*, mostly asian characters.
143 No listed terms, but visible copyright.
144 No **bold** or *italics*, mostly Chinese characters.
145 No listed terms, but visible copyright.
146 No **bold** or *italics*, mostly Chinese characters.
147 No listed terms, but visible copyright.
148 Entirely in **bold** and no *italics*, mostly Chinese characters.
149 No listed terms, but visible copyright.
150 No listed terms, but visible copyright.
151 Entirely in **bold**.
152 No listed terms, but visible copyright.
153 No **bold** or *italics*, mostly asian characters.
154 No listed terms, but visible copyright.

Font	No. of Glyphs	License	Origin
Adobe Ming Std[155]	18,156	Proprietary[156]	Adobe Systems Inc.
Free Serif	10,538	GPL 3.0+[157]	GNU Project
Quivira Regular	10,486	Proprietary[158]	Quivira-font.com
Symbola (version 9.17)	9,827	Free (permissive)[159]	George Douros
Everson Mono	9,756	Shareware	Michael Everson
Free Sans	6,272	GPL 3.0+[160]	GNU Project
Noto Sans Symbols[161]	5,356	SIL OFL (permissive)	Google
Apple Symbols	5,125	Proprietary[162]	Apple Inc.
Andika[163]	4,786	SIL OFL (permissive)	Gaultney, Olsen, et. al.
Free Mono	4,177	GPL 3.0+[164]	GNU Project
Times New Roman[165]	3,380	Proprietary[166]	Monotype Corporation
Tahoma[167]	3,301	Proprietary[168]	Microsoft Corporation
Code2001[169]	3,135	Shareware	James Kass
EB Garamond	3,084	SIL OFL (permissive)	Georg A. Duffner
Cormorant Garamond	2,821	SIL OFL (permissive)	Christian Thalmann
Cormorant	2,821	SIL OFL (permissive)	Christian Thalmann
Cormorant Infant	2,821	SIL OFL (permissive)	Christian Thalmann
CORMORANT SC	2,787	SIL OFL (permissive)	Christian Thalmann
CORMORANT UNICASE	2,787	SIL OFL (permissive)	Christian Thalmann
Garamond Premier Pro	2,735	Proprietary[170]	Adobe Systems Inc.

155 No **bold** or *italics*, mostly Chinese characters.
156 No listed terms, but visible copyright.
157 Includes the font embedding exception.
158 No listed terms, but visible copyright.
159 “Free for any use.” OTF version available here: http://files.au.adversary.org/files/fonts/Symbola.otf
160 Includes the font embedding exception.
161 The only font to correctly display glyphs for ₿ (U+20BF) and a number of other code points.
162 No listed terms, but visible copyright.
163 Additionally there are a number of related fonts providing coverage for various African and SE asian scripts.
164 Includes the font embedding exception.
165 Bundled with iBooks. May be common amongst Apple eReaders.
166 No listed terms, but visible copyright.
167 Included due to its use as the default font for complex layouts, including right-to-left languages such as Arabic and Hebrew.
168 No listed terms, but visible copyright.
169 A direct extension of Code2000. Note that this font is the most likely cause of “half-glyph” errors in most software using the Code2000 series of fonts. Disabling it in favour of Google's Noto fonts is advisable. Available here: http://files.au.adversary.org/files/fonts/CODE2001.ZIP
170 No listed terms, but visible copyright.

Font	No. of Glyphs	License	Origin
Twitter Color Emoji 1.3[171]	2,728	CC-BY-2.0	Brad Erickson
Seravek	2,596	Proprietary[172]	Process Type Foundry
Arimo[173]	2,584	Apache 2.0	Google
Tinos[174]	2,573	Apache 2.0	Google
Noto Sans	2,416	SIL OFL (permissive)	Google
Noto Serif	2,414	SIL OFL (permissive)	Google
Iowan Old Style	2,229	Proprietary[175]	Bitstream Inc.
Charter	2,092	Proprietary[176]	Bitstream Inc.
Palatino	1,756	Proprietary[177]	Linotype AG
Athelas	1,742	Proprietary[178]	TypeTogether
Minion Pro	1,682	Proprietary[179]	Adobe Systems Inc.
Adobe Arabic	1,508	Proprietary[180]	Adobe Systems Inc.
Times	1,443	Proprietary[181]	Apple Inc.[182]
Adobe Devangari[183]	1,316	Proprietary[184]	Adobe Systems Inc.
Cormorant Upright	1,216	SIL OFL (permissive)	Christian Thalmann
American Typewriter	1,195	Proprietary[185]	Int'l Typeface Corp.[186]
Adobe Naskh[187]	1,148	Proprietary[188]	Adobe Systems Inc.
Georgia	1,134	Proprietary[189]	Microsoft Corporation
Open Sans	936	Apache 2.0	Google
Open Sans Condensed	936	Apache 2.0	Google

171 See: https://github.com/eosrei/twemoji-color-font
172 Bundled with iBooks. May be common amongst Apple eReaders.
173 Google's answer to Arial.
174 Google's answer to Times New Roman.
175 Bundled with iBooks. May be common amongst Apple eReaders.
176 Bundled with iBooks. May be common amongst Apple eReaders.
177 Bundled with iBooks. May be common amongst Apple eReaders.
178 Bundled with iBooks. May be common amongst Apple eReaders.
179 No listed terms, but visible copyright.
180 No listed terms, but visible copyright.
181 No listed terms, but visible copyright.
182 Previously Linotype AG and Times is a registered trademark of Linotype AG. This appears to be a replacement for Times New Roman and is also listed with a PostScript name of Times Roman.
183 Mostly Hindi and Indian scripts.
184 No listed terms, but visible copyright.
185 No listed terms, but visible copyright.
186 The International Typeface Corporation (ITC).
187 Mostly Latin, Arabic and some asian. Medium weight only.
188 No listed terms, but visible copyright.
189 Bundled with iBooks and Kobo Aura H2O. May be common amongst Apple and Kobo eReaders and/or software.

Font	No. of Glyphs	License	Origin
Adobe Garamond Pro	935	Proprietary[190]	Adobe Systems Inc.
Latin Modern Roman	821	GUST Font License[191]	Polish TeX Users Group
Adobe Caslon Pro	801	Proprietary[192]	Adobe Systems Inc.
Adobe Jenson Pro	640	Proprietary[193]	Adobe Systems Inc.
Fairfield LT Std	429	Proprietary[194]	Adobe Systems Inc.
Garamond 3 LT Std	417	Proprietary[195]	Adobe Systems Inc.
Sabon LT Std	417	Proprietary[196]	Adobe Systems Inc.
Garamond MT Std	403	Proprietary[197]	Adobe Systems Inc.
Adobe Hebrew	396	Proprietary[198]	Adobe Systems Inc.
Sabon MT Std	324	Proprietary[199]	Adobe Systems Inc.
Inconsolata	303[200]	SIL OFL (permissive)	Raph Levien
Apple Braille	276	Proprietary[201]	Apple Inc.
ITC Veljovic Std	263	Proprietary[202]	Adobe[203] / ITC[204]
Plantin Std	253	Proprietary[205]	Adobe Systems Inc.
Unifont	57,089	GPL 2.0+[206]	GNU Project
Unifont CSUR	57,772	GPL 2.0+[207]	GNU Project
Unifont Upper	8,024	GPL 2.0+[208]	GNU Project
Unifont Upper CSUR	11,729	GPL 2.0+[209]	GNU Project

190 No listed terms, but visible copyright.
191 https://tug.org/fonts/licenses/GUST-FONT-LICENSE.txt
192 No listed terms, but visible copyright.
193 No listed terms, but visible copyright.
194 No listed terms, but visible copyright.
195 No listed terms, but visible copyright.
196 No listed terms, but visible copyright.
197 No listed terms, but visible copyright.
198 No listed terms, but visible copyright.
199 No listed terms, but visible copyright.
200 304 for the **bold** typeface and 359 for the Medium typeface.
201 No listed terms, but visible copyright.
202 No listed terms, but visible copyright.
203 Adobe Systems Inc.
204 International Typeface Corporation.
205 No listed terms, but visible copyright.
206 Includes the font embedding exception.
207 Includes the font embedding exception.
208 Includes the font embedding exception.
209 Includes the font embedding exception.

Frequently used characters and symbols

Alphabetic, Latin and Common Symbols

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
À	00c0	à	00e0	Ð	00d0	ð	00f0
Á	00c1	á	00e1	Ñ	00d1	ñ	00f1
Â	00c2	â	00e2	Ò	00d2	ò	00f2
Ã	00c3	ã	00e3	Ó	00d3	ó	00f3
Ä	00c4	ä	00e4	Ô	00d4	ô	00f4
Å	00c5	å	00e5	Õ	00d5	õ	00f5
Æ	00c6	æ	00e6	Ö	00d6	ö	00f6
Ç	00c7	ç	00e7	×	00d7	÷	00f7
È	00c8	è	00e8	Ø	00d8	ø	00f8
É	00c9	é	00e9	Ù	00d9	ù	00f9
Ê	00ca	ê	00ea	Ú	00da	ú	00fa
Ë	00cb	ë	00eb	Û	00db	û	00fb
Ì	00cc	ì	00ec	Ü	00dc	ü	00fc
Í	00cd	í	00ed	Ý	00dd	ý	00fd
Î	00ce	î	00ee	Þ	00de	þ	00fe
Ï	00cf	ï	00ef	ß	00df	ÿ	00ff
Ā	0100	ā	0101	Ē	0112	ē	0113
Ă	0102	ă	0103	Ī	012a	ī	012b
Ć	0106	ć	0107	Ō	014c	ō	014d
Ĉ	0108	ĉ	0109	Œ	0152	œ	0153
Č	010c	č	010d	Ş	015e	ş	015f
Ď	010e	ď	010f	Ū	016a	ū	016b
Ž	017d	ž	017e	Ʊ	01b1	Ƀ	0243
ɸ	0278	ʘ	0298				

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
ΐ	0390	Α	0391	Β	0392	Γ	0393
Δ	0394	Ε	0395	Ζ	0396	Η	0397
Θ	0398	Ι	0399	Κ	039a	Λ	039b
Μ	039c	Ν	039d	Ξ	039e	Ο	039f
Π	03a0	Ρ	03a1	Σ	03a3	Φ	03a6
Ψ	03a8	Ω	03a9	Ϊ	03aa	Ϋ	03ab
ά	03ac	έ	03ad	ή	03ae	ί	03af
ΰ	03b0	α	03b1	β	03b2	γ	03b3
δ	03b4	ε	03b5	ζ	03b6	η	03b7
θ	03b8	ι	03b9	κ	03ba	λ	03bb
μ	03bc	ν	03bd	ξ	03be	ο	03bf
π	03c0	ρ	03c1	ς	03c2	σ	03c3
τ	03c4	υ	03c5	φ	03c6	χ	03c7
ψ	03c8	ω	03c9	ϊ	03ca	ϋ	03cb
ό	03cc	ύ	03cd	ώ	03ce	Ϗ	03cf

Cyrillic Characters

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
Ѐ	0400[210]	Ё	0401	Ѕ	0405	І	0406
Ї	0407	Ј	0408	А	0410	В	0412
Е	0415	Ж	0416	З	0417	К	041a
М	041c	Н	041d	О	041e	П	041f
Р	0420	С	0421	Т	0422	У	0423
Ф	0424	Х	0425	Ь	042c	Э	042e

210 The code points between 0x0400 and 0x04ff are cyrillic characters, but include multiple characters which bear a strong resemblance to certain ASCII characters without being those characters. Which means these are the characters most frequently employed to masquerade as a Latin characters as components of phishing sites.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
Я	042f	а	0430	в	0432	е	0435
ж	0436	з	0437	к	043a	м	043c
н	043d	о	043e	р	0440	с	0441
т	0442	у	0443	х	0445	ѐ	0450
ё	0451	є	0454	ѕ	0455	і	0456
ї	0457	ј	0458	љ	0459	ќ	045c

Mathematical Alpha-Numeric Characters

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
𝐀	1d400[211]	𝐚	1d41a	𝐴	1d434	𝑎	1d44e
𝑨	1d468	𝒂	1d482	𝒜	1d49c	𝒶	1d4b6
𝓐	1d4d0	𝓩	1d4e9	𝓪	1d4ea	𝔃	1d503
𝔄	1d504	𝔞	1d51e	𝔸	1d538	𝕒	1d552
𝕬	1d56c	𝖆	1d586	𝖠	1d5a0	𝖺	1d5ba
𝗔	1d5d4	𝗮	1d5ee	𝘈	1d608	𝘢	1d622
𝘼	1d63c	𝙖	1d656	𝙰	1d670	𝚊	1d68a[212]
𝛂	1d6c2	𝛢	1d6e2	𝛼	1d6fc	𝜜	1d71c
𝜶	1d736	𝝖	1d756	𝝰	1d770	𝞐	1d790
𝞪	1d7aa	𝞻	1d7bb	𝟎	1d7ce[213]	𝟘	1d7d8
𝟢	1d7e2	𝟬	1d7ec	𝟶	1d7f6	𝟿	1d7ff

211 The beginning of a range of mathematical symbols, from 1d400 through to 1d7ff which also provide deceptive alternative text characters which will be read as plain latin characters (1d400 to 1d6a7), Greek letters (1d6a8 to 1d7cd), or as numerals or digits. A small number of characters within the alphabet types are missing or not formally assigned yet. Only the first letter of each alphabet set (i.e. A or Alpha) are included in these charts.
212 The first of the Greek alphabet sets.
213 The first of the numeric digit sets.

Deceptive Possible Punycode

Most of these examples demonstrate possible spoofed gTLDs along with a handful of possible subdomains or 2LDs.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
.net[214]	.com[215]	.org[216]	.sex[217]	.xxx[218]	.edu[219]	.art[220]	.gov[221]
.info[222]	.info[223]	.sci[224]	.huЬ[225]	.huЬ .sci[226]	.sci. huЬ[227]	.sci huЬ[228]	.sci-huЬ[229]

214 Spoofed version of `.net` that would really be `xn—nt-nlc`.
215 Spoofed version of `.com` using two replacement characters and would really be `xn—m-0tbi`. Alternatively the two versions each replacing one character are `xn--om-nmc` and `xn—cm-fmc`.
216 Spoofed version of .org that would really be `xn--rg-emc`.
217 The .sex domain is still just a proposal, but if it passes this demonstrates that spoofing it will have multiple options and this is the most complex one, so it resulted in `xn--e1a6a7b`.
218 The .xxx domain does exist and yet has numerous options for spoofing. The current effort resulted in this `xn--u1aaa`.
219 Spoofed version of .edu that would really be `xn--du-mlc`.
220 Spoofed version of .art that would really be `xn--rt-6kc`.
221 Spoofed version of .gov that would really be `xn--gv-fmc`.
222 Spoofed version of .info that would really be `xn--nf-gmc7g`.
223 Spoofed version of .info that would really be `xn--inf-ued`.
224 Spoofed version of .sci that would really be `xn--q1a3be`.
225 Spoofed version of .hub that would really be `xn--hu-nnc`.
226 Spoofed version of .hub.sci that would really be `xn--hu-nnc.xn--q1a3be`.
227 Spoofed version of .sci.hub that would really be `xn—q1a3be.xn--hu-nnc`.
228 Spoofed version of .scihub that would really be `xn--hu-nmc0c0bh`.
229 Spoofed version of .sci-hub that would really be `xn--hu-3ed2d6bi`.

Punctuation

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
"	0022[230]	'	0027[231]	-	002d[232]	_	005f[233]
`	0060[234]	|	007c[235]	´	00b4	·	00b7[236]
ʹ	02b9	ʺ	02ba	ʻ	02bb	ʼ	02bc
ʽ	02bd			ˆ	02c6	ˇ	02c7
ˈ	02c8	ˉ	02c9	ˊ	02ca	ˋ	02cb
ˌ	02cc	ˍ	02cd	ˎ	02ce	ˏ	02cf
‐	2010[237]	‑	2011[238]	‒	2012[239]	–	2013[240]
—	2014[241]	―	2015	‖	2016	‗	2017
‘	2018[242]	’	2019[243]	‚	201a	‛	201b
“	201c[244]	”	201d[245]	„	201e	‟	201f
†	2020	‡	2021	•	2022	‣	2023
․	2024	‥	2025	…	2026	‧	2027
	2060[246]						

230 On the keyboard (QUOTATION MARK, via Shift + APOSTROPHE), note that OS X automatically turns this into either 0x201c (Alt + [) or 0x201d (Alt + Shift + [) depending on what context it attempted to discern the key was being used. This is done in LibreOffice, but not in Emacs. This is important to note if I belt out some sample dialogue in my text editor and need to incorporate it into a book later.
231 On the keyboard (the APOSTROPHE), note that this character is not transformed into the curly equivalent by OS X, so those marks need to be accessed by the OS X alternative keys or via their code points.
232 This is the HYPHEN-MINUS.
233 This is the UNDERSCORE / SPACING UNDERSCORE / LOW LINE.
234 This is the GRAVE ACCENT adjacent to the "1" at the top left of the keyboard.
235 This is the PIPE / VERTICAL LINE / VERTICAL BAR.
236 Middle dot or decimal point, really good for using in place of a full stop on sites like twitter.com where domain names are automatically expanded to include the protocol and a link and you lose available characters for posting (or might not want an active link). 0x22c5 can also be used for this purpose.
237 This is the HYPHEN. Not to be confused with the HYPHEN-MINUS (0x002d) on the keyboard.
238 This is the NON-BREAKING HYPHEN.
239 This is the FIGURE DASH.
240 This is the EN-DASH.
241 This is the EM-DASH.
242 OS X shortcut: Alt +]
243 OS X shortcut: Alt + Shift +]
244 OS X shortcut: Alt + [
245 OS X shortcut: Alt + Shift + [
246 This is the NON-BREAKING SPACE. Very good for making fediverse and/or Twitter @name style usernames appear as they should without linking to the account in question.

Currency and Common Symbols

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
≠	2260	¿	00bf	¡	00a1	¢	00a2
£	00a3	¤	00a4	¥	00a5	¦	00a6
§	00a7	¨	00a8	©	00a9	ª	00aa
«	00ab	»	00bb	®	00ae	¯	00af
°	00b0	±	00b1	²	00b2	³	00b3
º	00ba	¶	00b6	¸	00b8	¹	00b9
₠	20a0	₡	20a1	₢	20a2	₣	20a3
₤	20a4	₥	20a5	₦	20a6	₧	20a7
₨	20a8	₩	20a9	₪	20aa	₫	20ab
€	20ac	₭	20ad	₮	20ae	₯	20af
₰	20b0	₱	20b1	₲	20b2	₳	20b3
₴	20b4	₵	20b5	₶	20b6	₷	20b7
₸	20b8	₹	20b9	₺	20ba	₼	20bc
₽	20bd	₽	20bd	₾	20be	₿[247]	20bf
⃦	20e6[248]	◌̸	0338[249]	฿	0e3f	№	2116
℗	2117	℠	2120	℡	2121	™	2122

247 Note that this Bitcoin logo symbol is not what appears in all fonts. It works with Google's Noto Sans Symbols, but displays a different character with Times New Roman and some other fonts. It is unclear why there appears to be a character conflict at this stage. The Bitcoin symbol was added with the updates in Unicode 10.0.

248 Overlay character.

249 Overlay character.

Direction

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
←	2190	↑	2191	→	2192	↓	2193
↔	2194	↕	2195	↖	2196	↗	2197
↘	2198	↙	2199	↚	219a	↛	219b
↜	219c	↝	219d	↞	219e	↟	219f
↠	21a0	↡	21a1	↢	21a2	↣	21a3
↤	21a4	↥	21a5	↦	21a6	↧	21a7
↨	21a8	↩	21a9	↪	21aa	↫	21ab
↬	21ac	↭	21ad	↮	21ae	↯	21af
⇄	21c4	⇅	21c5	⇆	21c6	⇇	21c7
⇈	21c8	⇉	21c9	⇊	21ca	⇋	21cb
⇌	21cc	⇍	21cd	⇎	21ce	⇏	21cf
❛	275b	❜	275c	❝	275d	❞	275e
❟	275f	❠	2760	➔	2794	➙	2799
➛	279b	➜	279c	➝	279d	➞	279e
➟	279f	➤	27a4	➩	27a9	➾	27be

Fractions and Numbering

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
¼	00bc	½	00bd	¾	00be	⅐	2150
⅑	2151	⅒	2152	⅓	2153	⅔	2154
⅕	2155	⅖	2156	⅗	2157	⅘	2158
⅙	2159	⅚	215a	⅛	215b	⅜	215c
⅝	215d	⅞	215e	⅟	215f		
①	2460	⑳	2473	⑴	2474	⒇	2487
⒈	2488	⒛	249b	⒜	249c	⒵	24b5
Ⓐ	24b6	Ⓩ	24cf	ⓐ	24d0	ⓩ	24e9
⓪	24ea	⓫	24eb	⓴	24f4	⓵	24f5
⓶	24f6	⓽	24fd	⓾	24fe	⓿	24ff

Sex, Sexuality and Gender

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
☿	263f	♀	2640	♁	2641	♂	2642
⚢	26a2	⚣	26a3	⚤	26a4	⚥	26a5
⚦	26a6	⚧	26a7	⚨	26a8	⚩	26a9
⚭	26ad	⚮	26ae	⚯	26af	⚲	26b2

Various Symbols – Plane 0

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
ॐ	0950	†	2020[250]	‡	2021[251]	Ω	2126
℧	2127	∑	2211	−	2212[252]	∓	2213
∔	2214	∕	2215	∖	2216	∗	2217
∘	2218	∙	2219	√	221a	∛	221b
∜	221c	∝	221d	∞	221e	∟	221f
∴	2234	∵	2235	∶	2236	∷	2237
∸	2238	∹	2239	∺	223a	∻	223b
∼	223c	∽	223d	∾	223e	∿	223f
⊔	2294	⊕	2295	⊖	2296	⊗	2297
⊘	2298	⊙	2299	⊚	229a	⊛	229b
⊜	229c	⊝	229d	⊞	229e	⊟	229f
⊠	22a0	⊡	22a1	⊰	22b0	⊱	22b1
⊲	22b2	⊳	22b3	⊴	22b4	⊵	22b5
⋄	22c4	⋅	22c5	⋆	22c6	⋇	22c7
⋬	22ec	⋭	22ed	⋮	22ee	⋯	22ef
⌔	2314	⌕	2315	⌖	2316	⌗	2317
⌘	2318	⌚	231a	⌛	231b	⍎	234e
⍏	234f	⍐	2350	⍓	2353	⍕	2355
⍖	2356	⍗	2357	⍙	2359	⍚	235a
⍛	235b	⍜	235c	⍝	235d	⍞	235e
⍟	235f	⍦	2366	⍯	236f	⍰	2370
☀	2600	☁	2601	☂	2602	☃	2603

250 This version of the DAGGER is only available with the Cormorant fonts and only active by default with Cormorant Infant.

251 This rapier symbol is the DOUBLE DAGGER; it is only available with the Cormorant fonts and only active by default with Cormorant Infant.

252 This is the actual MINUS SIGN as distinct from a hyphen, figure dash, en-dash or em-dash (see above).

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
☄	2604	★	2605	☆	2606	☇	2607
☈	2608	☉	2609	☊	260a	☋	260b
☌	260c	☍	260d	☎	260e	☏	260f
☐	2610	☑	2611	☒	2612	☓	2613
☠	2620	☡	2621	☢	2622	☣	2623
☤	2624	☥	2625	☦	2626	☧	2627
☨	2628	☩	2629	☪	262a	☫	262b
☬	262c	☭	262d	☮	262e	☯	262f
☸	2638	☹	2639	☺	263a	☻	263b
☼	263c	☽	263d	☾	263e	☿	263f[253]
♀	2640[254]	♁	2641[255]	♂	2642[256]	♃	2643
♄	2644	♅	2645	♆	2646	♇	2647
♈	2648	♉	2649	♊	264a	♋	264b
♌	264c	♍	264d	♎	264e	♏	264f
♐	2650	♑	2651	♒	2652	♓	2653
♔	2654	♕	2655	♖	2656	♗	2657
♘	2658	♙	2659	♚	265a	♛	265b
♜	265c	♝	265d	♞	265e	♟	265f
♠	2660	♡	2661	♢	2662	♣	2663
♤	2664	♥	2665	♦	2666	♧	2667

253 This is Mercury and is frequently used to denote intersex (because Mercury was Hermes and Hermes was the father of Hermaphroditus).
254 The female sign, also used for Venus and thus Aphodite.
255 The Earth sign.
256 The male sign, also used for Mars.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
♨	2668	♩	2669	♪	266a	♫	266b
♬	266c	♭	266d	♮	266e	♯	266f
♰	2670	♱	2671	♲	2672	♳	2673
♴	2674	♹	2679	♺	267a	♻	267b
♼	267c	♽	267d	♾	267e	♿	267f
⚐	2690	⚑	2691	⚒	2692	⚓	2693
⚔	2694	⚕	2695	⚖	2696	⚗	2697
⚘	2698	⚙	2699	⚚	269a	⚛	269b
⚜	269c	⚝	269d	⚞	269e	⚟	269f
⚠	26a0	⚡	26a1	⚢	26a2[257]	⚣	26a3[258]
⚤	26a4[259]	⚥	26a5[260]	⚦	26a6[261]	⚧	26a7[262]
⚨	26a8	⚩	26a9	⚪	26aa	⚫	26ab
⚬	26ac	⚭	26ad	⚮	26ae	⚯	26af
⚰	26b0	⚱	26b1	⚲	26b2	⚳	26b3
⚴	26b4	⚵	26b5	⚶	26b6	⚷	26b7
⚸	26b8	⚹	26b9	⚺	26ba	⚻	26bb
⚼	26bc	⚽	26bd	⚾	26be	⚿	26bf
⛠	26e0	⛡	26e1	⛢	26e2	⛣	26e3

257 The doubled female sign.
258 The doubled male sign.
259 The interlocked male and female sign.
260 The male and female sign. Humans or binary male/female representation of humanity.
261 The male with stroke sign.
262 The male with stroke and male and female sign. Adopted to designate transgender and transsexual or other non-binary gender groups.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
⛤	26e4[263]	⛥	26e5[264]	⛦	26e6[265]	⛧	26e7[266]
⛨	26e8	⛩	26e9	⛪	26ea	⛫	26eb
⛬	26ec	⛭	26ed	⛮	26ee	⛯	26ef
⛰	26f0	⛱	26f1	⛲	26f2	⛳	26f3
⛴	26f4	⛵	26f5	⛶	26f6	⛷	26f7
⛸	26f8	⛹	26f9	⛺	26fa	⛻	26fb
⛼	26fc	⛽	26fd	⛾	26fe	⛿	26ff
✀	2700	✁	2701	✂	2702	✃	2703
✠	2720	✡	2721	✢	2722	✣	2723
✤	2724	✥	2725	✦	2726	✧	2727
✨	2728	✩	2729	✪	272a	✫	272b
✬	272c	✭	272d	✮	272e	✯	272f
✰	2730	✱	2731	✲	2732	✳	2733
⟼	27fc	⟽	27fd	⟾	27fe	⟿	27ff
⠀	2800[267]	⣿	28ff[268]	⤀	2900	⨀	2a00
⬀	2b00	⬛	2b1b	⬜	2b1c	⯯	2bef
Ⱡ	2c60[269]	Ɐ	2c6f	Ɒ	2c70	Ɀ	2c7f
一	4e00[270]	卍	534d[271]	卐	5350[272]	鿪	9fea[273]

263 The PENTAGRAM.
264 The RIGHT-HANDED INTERLACED PENTAGRAM.
265 The LEFT-HANDED INTERLACED PENTAGRAM.
266 The INVERTED PENTAGRAM.
267 Braille is from 2800 to 28ff and glyph support is uncommon. Code2000 is used here.
268 In this case, a fallback font is utilised instead of Code2000.
269 This line is Latin Extended-C.
270 Beginning of CJK Unified Ideographs. Largely only noted due to the following two characters included in that range and the occasional others noted further down (ref.: taipan, *I Ching* and *shi*).
271 The LEFT SWASTIKA sign.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
◌︀	fe00[274]	◌︍	fe0d[275]	◌︎	fe0e[276]	◌️	fe0f[277]
！	ff01	＃	ff03	＄	ff04	％	ff05
＆	ff06	＊	ff0a	￠	ffe0	￡	ffe1
￥	ffe5	￭	ffed	￮	ffee	�	fffd[278]

272 The RIGHT SWASTIKA sign. This is the version or direction of the SWASTIKA used by the Nazi Party in Germany (noting this because I always get them mixed up and forget).
273 End of the CJK Unified Ideographs.
274 Variation selector.
275 Variation selector.
276 Emoji variation selector.
277 Emoji variation selector.
278 The end of 3 byte UTF-8 characters. Neither `0xfffe` nor `0xffff` are assigned glyphs.

Various Symbols – Plane 1

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
𓁚	1305a[279]	𓁛	1305b[280]	𓁜	1305c[281]	𓁝	1305d[282]
𓁞	1305e[283]	𓁟	1305f[284]	𓁠	13060[285]	𓁡	13061[286]
𓁢	13062[287]	𓁣	13063[288]	𓁤	13064[289]	𓁥	13065[290]
𓁦	13066[291]	𓁧	13067[292]	𓁨	13068[293]	𓁩	13069[294]
𓁪	1306a[295]	𓁫	1306b	𓁬	1306c	𓁭	1306d
𓁮	1306e[296]	𓁯	1306f[297]	𓁰	13070[298]	𓁱	13071[299]
𓁲	13072[300]	𓁳	13073[301]	𓁴	13074[302]	𓁵	13075[303]
𓂀	13080[304]	𓃠	130e0[305]	𓃡	130e1[306]	𓃢	130e2[307]
𓅃	13143[308]	𓅝	1315d[309]	𓅞	1315e[310]	𓉴	13274[311]

279 Egyptian hieroglyph for the god Ra.
280 Egyptian hieroglyph for the god Ra with ankh.
281 Egyptian hieroglyph for the god Thoth with falcon head.
282 Egyptian hieroglyph for the god Thoth with falcon head (reversed).
283 Egyptian hieroglyph for the god Ra with ankh (reversed).
284 Egyptian hieroglyph for the god Thoth with ibis head.
285 Egyptian hieroglyph for the god Chnum.
286 Egyptian hieroglyph for the god Chnum with ankh.
287 Egyptian hieroglyph for the god Anubis (or Wepwawet if blue).
288 Egyptian hieroglyph for the god Set.
289 Egyptian hieroglyph for the god Min.
290 Egyptian hieroglyph for the goddess Hathor.
291 Egyptian hieroglyph for the goddess Ma'at.
292 Egyptian hieroglyph for the goddess Ma'at with ankh.
293 Egyptian hieroglyph for the god Hah.
294 Egyptian hieroglyph for the god Amun.
295 Egyptian hieroglyph for the god Amun: (reversed); with scimitar; with scimitar (reversed); with ref crown and ankh.
296 Egyptian hieroglyph for the god Month.
297 Egyptian hieroglyph for the god Tatenen.
298 Egyptian hieroglyph for the god Ptah.
299 Egyptian hieroglyph for the god Ptah in shrine.
300 Egyptian hieroglyph for the god Bes.
301 Egyptian hieroglyph for the god Thoth with falcon head and moon.
302 Egyptian hieroglyph for the goddess with feline head and sun (possibly Bastet / Bubastis).
303 Egyptian hieroglyph for the god Amun with red crown and scepter.
304 Egyptian hieroglyph for the Eye of Horus.
305 Egyptian hieroglyph for a cat.
306 Egyptian hieroglyph for a dog.
307 Egyptian hieroglyph for a supine canine (i.e. a lying dog; ref. Australian slang for criminal informant).
308 Egyptian hieroglyph for a falcon (Horus).
309 Egyptian hieroglyph for the sacred ibis on standard (Thoth).
310 Egyptian hieroglyph for the sacred ibis (Thoth).
311 Egyptian hieroglyph for a pyramid.

Emojis

Due to problems with LibreOffice using extended fonts, including the colour Emoji fonts and exporting them to PDF, the Emoji cheat sheet has been moved to a separate file.[312] hopefully this will be temporary, but given the nature and status of the bug which causes it, it is likely to take a while before it is fixed.[313]

A small subset of non-colour fallback Emoji are included below. This includes the letters which are used as country codes to display the relevant flags of those countries on some platforms (e.g. Twitter and Signal). Additional characters can be checked in the Unicode code charts and online at Emojipedia.[314]

The following two sections, however, use a custom font, Twitter Color Emoji,[315] which is a TTF build from the Twitter Emoji sources,[316] and which includes a black and white fallback which will display in LibreOffice and thus render in a PDF. In the unlikely event these are ever needed in an EPUB, then the Twitter Color Emoji TTF file will be converted to OTF and embedded.

There is partial duplication for the section on using ISO 3166-1 Alpha-2[317] country codes to display Emoji flags. Note that some codes might not be configured and some older states or nation-states do not produce the old flags. For instance the old code for the Soviet Union (SU) no longer produces the Soviet flag. Some other international flags, such as the NATO flag, are also unavailable as NATO lacks a two character code and reserved codes, like those beginning with the letter X, are not coded in the Unicode standard.

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
	fe0f[318]	🀄	1f004	🃏	1f0cf	🅰	1f170
🅱	1f171	🅾	1f17e	🅿	1f17f	🆎	1f18e
🌈	1f308	🤦	1f926				

312 EmojiUnicodeNotes.odt.
313 https://bugs.documentfoundation.org/show_bug.cgi?id=94655
314 https://emojipedia.org/objects/
315 https://github.com/eosrei/twemoji-color-font
316 https://github.com/twitter/twemoji
317 https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
318 Variation Selector-16. Used when a glyph defaults to text presentation to indicate that the glyph should use an emoji presentation. Used immediately following the glyph to be specified.

Twitter Color Emojis

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
🆑	1f191	🆒	1f192	🆓	1f193	🆔	1f194
🆕	1f195	🆖	1f196	🆗	1f197	🆘	1f198
🆙	1f199	🆚	1f19a	🇦	1f1e6	🇧	1f1e7
🇨	1f1e8	🇩	1f1e9	🇪	1f1ea	🇫	1f1eb
🇬	1f1ec	🇭	1f1ed	🇮	1f1ee	🇯	1f1ef
🇰	1f1f0	🇱	1f1f1	🇲	1f1f2	🇳	1f1f3
🇴	1f1f4	🇵	1f1f5	🇶	1f1f6	🇷	1f1f7
🇸	1f1f8	🇹	1f1f9	🇺	1f1fa	🇻	1f1fb
🇼	1f1fc	🇽	1f1fd	🇾	1f1fe	🇿	1f1ff
🌀	1f300	🌁	1f301	🌂	1f302	🌃	1f303
🌄	1f304	🌅	1f305	🌆	1f306	🌇	1f307
🌈	1f308	🌉	1f309	🌊	1f30a	🌋	1f30b
🌌	1f30c	🌍	1f30d	🌎	1f30e	🌏	1f30f
🌐	1f310	🌑	1f311	🌒	1f312	🌓	1f313
🌔	1f314	🌕	1f315	🌖	1f316	🌗	1f317
🌘	1f318	🌙	1f319	🌚	1f31a	🌛	1f31b
🌜	1f31c	🌝	1f31d	🌞	1f31e	🌟	1f31f
🌠	1f320	🌡	1f321	🌤	1f324	🌥	1f325
🌦	1f326	🌧	1f327	🌨	1f328	🌩	1f329
🌪	1f32a	🌫	1f32b	🌱	1f331	🌲	1f332
🌳	1f333	🌴	1f334	🌵	1f335	🌶	1f336
🌷	1f337	🌹	1f339	🌻	1f33b	🌼	1f33c
🍀	1f340	🍄	1f344	🍵	1f375	🍸	1f378
🎀	1f380	🎁	1f381	🎂	1f382	🎃	1f383
🎄	1f384	🎅	1f385	🎆	1f386	🎇	1f387
🐀	1f400	🐁	1f401	🐂	1f402	🐃	1f403

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
🐄	1f404	🐅	1f405	🐆	1f406	🐇	1f407
🐈	1f408	🐉	1f409	🐊	1f40a	🐋	1f40b
🐌	1f40c	🐍	1f40d	🐎	1f40e	🐏	1f40f
🐐	1f410	🐑	1f411	🐒	1f412	🐓	1f413
🐔	1f414	🐕	1f415	🐖	1f416	🐗	1f417
🐘	1f418	🐙	1f419	🐚	1f41a	🐛	1f41b
🐜	1f41c	🐝	1f41d	🐞	1f41e	🐟	1f41f
🐠	1f420	🐡	1f421	🐢	1f422	🐣	1f423
🐤	1f424	🐥	1f425	🐦	1f426	🐧	1f427
🐨	1f428	🐩	1f429	🐪	1f42a	🐫	1f42b
🐬	1f42c	🐭	1f42d	🐮	1f42e	🐯	1f42f
🐰	1f430	🐱	1f431	🐲	1f432	🐳	1f433
🐴	1f434	🐵	1f435	🐶	1f436	🐷	1f437
🐸	1f438	🐹	1f439	🐺	1f43a	🐻	1f43b
🐼	1f43c	🐽	1f43d	🐾	1f43e	🐿	1f43f
👀	1f440	👁	1f441	👂	1f442	👃	1f443
👄	1f444	👅	1f445	👆	1f446	👇	1f447
👈	1f448	👉	1f449	👊	1f44a	👋	1f44b
👌	1f44c	👍	1f44d	👎	1f44e	👏	1f44f
👸	1f478	👹	1f479	👺	1f47a	👻	1f47b
👼	1f47c	👽	1f47d	👾	1f47e	👿	1f47f
💀	1f480	💁	1f481	💂	1f482	💃	1f483
💈	1f488	💉	1f489	💊	1f48a	💋	1f48b
💌	1f48c	💍	1f48d	💎	1f48e	💏	1f48f
💐	1f490	💑	1f491	💒	1f492	💓	1f493
💔	1f494	💕	1f495	💖	1f496	💗	1f497
💘	1f498	💙	1f499	💚	1f49a	💛	1f49b

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
💜	1f49c	💝	1f49d	💞	1f49e	💟	1f49f
💠	1f4a0	💡	1f4a1	💢	1f4a2	💣	1f4a3
💤	1f4a4	💥	1f4a5	💦	1f4a6	💧	1f4a7
💨	1f4a8	💩	1f4a9	💪	1f4aa	💫	1f4ab
💬	1f4ac	💭	1f4ad	💮	1f4ae	💯	1f4af
🔐	1f510	🔑	1f511	🔒	1f512	🔓	1f513
🔠	1f520	🔡	1f521	🔢	1f522	🔣	1f523
🔤	1f524	🔥	1f525	🔦	1f526	🔧	1f527
🔨	1f528	🔩	1f529	🔪	1f52a	🔫	1f52b
🔬	1f52c	🔭	1f52d	🔮	1f52e	🔯	1f52f
🕵	1f575	🗝	1f5dd	🗡	1f5e1	🗣	1f5e3
😀	1f600	😁	1f601	😂	1f602	😃	1f603
😄	1f604	😅	1f605	😆	1f606	😇	1f607
😈	1f608	😉	1f609	😊	1f60a	😋	1f60b
😌	1f60c	😍	1f60d	😎	1f60e	😏	1f60f
😐	1f610	😑	1f611	😒	1f612	😓	1f613
😔	1f614	😕	1f615	😖	1f616	😗	1f617
😘	1f618	😙	1f619	😚	1f61a	😛	1f61b
😜	1f61c	😝	1f61d	😞	1f61e	😟	1f61f
😠	1f620	😡	1f621	😢	1f622	😣	1f623
😤	1f624	😥	1f625	😦	1f626	😧	1f627
😨	1f628	😩	1f629	😪	1f62a	😫	1f62b
😬	1f62c	😭	1f62d	😮	1f62e	😯	1f62f
😰	1f630	😱	1f631	😲	1f632	😳	1f633
😴	1f634	😵	1f635	😶	1f636	😷	1f637
😸	1f638	😹	1f639	😺	1f63a	😻	1f63b
😼	1f63c	😽	1f63d	😾	1f63e	😿	1f63f

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
🙀	1f640	🙁	1f641	🙂	1f642	🙃	1f643
🙄	1f644	🙅	1f645	🙆	1f646	🙇	1f647
🙈	1f648	🙉	1f649	🙊	1f64a	🙋	1f64b
🙌	1f64c	🙍	1f64d	🙎	1f64e	🙏	1f64f
🚀	1f680	🚁	1f681	🚂	1f682	🚫	1f6ab
🚬	1f6ac	🚭	1f60d	🚮	1f6ae	🚯	1f6af
🚰	1f6b0	🚱	1f6b1	🚲	1f6b2	🚳	1f6b3
🚴	1f6b4	🚵	1f6b5	🚶	1f6b6	🚷	1f6b7
🤐	1f910	🤑	1f911	🤒	1f912	🤓	1f913
🤔	1f914	🤕	1f915	🤖	1f916	🤗	1f917
🤘	1f918	🤙	1f919	🤚	1f91a	🤛	1f91b
🤜	1f91c	🤝	1f91d	🤞	1f91e	🤟	1f91f
🤠	1f920	🤡	1f921	🤢	1f922	🤣	1f923
🤤	1f924	🤥	1f925	🤦	1f926	🤧	1f927
🤨	1f928	🤩	1f929	🤪	1f92a	🤫	1f92b
🤬	1f92c	🤭	1f92d	🤮	1f92e	🤯	1f92f
🤺	1f93a	🥂	1f942	🥊	1f94a	🥋	1f94b
🦀	1f980	🦁	1f981	🦂	1f982	🦃	1f983
🦄	1f984	🦅	1f985	🦆	1f986	🦇	1f987
🦈	1f988	🦉	1f989	🦊	1f98a	🦋	1f98b
🦌	1f98c	🦍	1f98d	🦎	1f98e	🦏	1f98f
🦕	1f995	🦖	1f996	🧐	1f9d0	🧙	1f9d9
🧚	1f9da	🧛	1f9db	🧜	1f9dc	🧝	1f9dd
🧞	1f9de	🧟	1f9df	🧠	1f9e0	🜚	1f71a

Twitter Color Emoji Country Code Flags

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
🇦	1f1e6	🇧	1f1e7	🇨	1f1e8	🇩	1f1e9
🇪	1f1ea	🇫	1f1eb	🇬	1f1ec	🇭	1f1ed
🇮	1f1ee	🇯	1f1ef	🇰	1f1f0	🇱	1f1f1
🇲	1f1f2	🇳	1f1f3	🇴	1f1f4	🇵	1f1f5
🇶	1f1f6	🇷	1f1f7	🇸	1f1f8	🇹	1f1f9
🇺	1f1fa	🇻	1f1fb	🇼	1f1fc	🇽	1f1fd
🇾	1f1fe	🇿	1f1ff				

Most Frequently Used Flags

Flag	C-Code	Flag	C-Code	Flag	C-Code	Flag	C-Code
🇦🇪	AE	🇦🇫	AF	🇦🇶	AQ	🇦🇷	AR
🇦🇺	AU	🇧🇪	BE	🇨🇦	CA	🇨🇱	CL
🇨🇳	CN	🇨🇴	CO	🇨🇺	CU	🇩🇪	DE
🇪🇬	EG	🇪🇸	ES	🇪🇺	EU	🇫🇮	FI
🇫🇷	FR	🇬🇧	GB	🇬🇷	GR	🇭🇰	HK
🇮🇩	ID	🇮🇪	IE	🇮🇱	IL	🇮🇲	IM
🇮🇳	IN	🇮🇴	IO	🇮🇶	IQ	🇮🇷	IR
🇮🇸	IS	🇮🇹	IT	🇯🇵	JP	🇰🇵	KP
🇰🇷	KR	🇰🇼	KW	🇱🇧	LB	🇲🇨	MC
🇲🇴	MO	🇲🇹	MT	🇲🇽	MX	🇳🇴	NO
🇳🇷	NR	🇳🇿	NZ	🇵🇬	PG	🇵🇷	PR
🇵🇸	PS	🇷🇴	RO	🇷🇺	RU	🇸🇦	SA
🇸🇪	SE	🇸🇬	SG	🇹🇱	TL	🇹🇷	TR
🇹🇼	TW	🇺🇳	UN	🇺🇸	US	🇿🇦	ZA

Chess, CJK and the Rest of Asia

Text Chess Diagrams

Though it is a bit tricky to pull off in some mediums, like Twitter and Mastodon, entire chess boards can be made with Unicode characters. In addition to the chess pieces in `0x2654` to `0x265f`, the squares of the board are made with `0x2b1b` and `0x2b1c`.

The two examples on the left use Times New Roman figures and and Symbola squares or a possible default colour replacement. Size matching is performed by setting the figures to 24 point font size and the squares to 22 points. Then the sizes are adjusted to equal relative values. The examples below are at half size with 12 point chess figures and 11 point squares of the board.

The two examples on the right use Cormorant Garamond for both the figurines and the squares. The size matching is performed with 19 point font size for the figurines and 22 point fonts for the squares of the board. Then those sizes are adjusted to relative equivalent values. The two examples below are approximately 63% to 65% of that size with 12 point figures and 13.9 point squares.

♜♞♝♛♚♝♞♜
♙♙♙♙♙♙♙♙
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
♙♙♙♙♙♙♙♙
♖♘♗♕♔♗♘♖

♖♘♗♔♕♗♘♖
♙♙♙♙♙♙♙♙
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
♙♙♙♙♙♙♙♙
♜♞♝♚♛♝♞♜

♜♞♝♛♚♝♞♜
♙♙♙♙♙♙♙♙
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
♙♙♙♙♙♙♙♙
♖♘♗♕♔♗♘♖

♖♘♗♔♕♗♘♖
♙♙♙♙♙♙♙♙
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
⬜⬛⬜⬛⬜⬛⬜⬛
⬛⬜⬛⬜⬛⬜⬛⬜
♙♙♙♙♙♙♙♙
♜♞♝♚♛♝♞♜

I Ching lines, trigrams and characters

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
☰	2630	☴	2634	⚊	268a	⚋	268b
☱	2631	☵	2635	⚌	268c	⚍	268d
☲	2632	☶	2636	⚎	268e	⚏	268f
☳	2633	☷	2637	⿇	2fc7	大	5927
易	6613	死	6b7b	經	7d93	经	7ecf
班	73ed						

I Ching hexagrams

Character	Hex Code	Character	Hex Code	Character	Hex Code	Character	Hex Code
䷀	4dc0	䷐	4dd0	䷠	4de0	䷰	4df0
䷁	4dc1	䷑	4dd1	䷡	4de1	䷱	4df1
䷂	4dc2	䷒	4dd2	䷢	4de2	䷲	4df2
䷃	4dc3	䷓	4dd3	䷣	4de3	䷳	4df3
䷄	4dc4	䷔	4dd4	䷤	4de4	䷴	4df4
䷅	4dc5	䷕	4dd5	䷥	4de5	䷵	4df5
䷆	4dc6	䷖	4dd6	䷦	4de6	䷶	4df6
䷇	4dc7	䷗	4dd7	䷧	4de7	䷷	4df7
䷈	4dc8	䷘	4dd8	䷨	4de8	䷸	4df8
䷉	4dc9	䷙	4dd9	䷩	4de9	䷹	4df9
䷊	4dca	䷚	4dda	䷪	4dea	䷺	4dfa
䷋	4dcb	䷛	4ddb	䷫	4deb	䷻	4dfb
䷌	4dcc	䷜	4ddc	䷬	4dec	䷼	4dfc
䷍	4dcd	䷝	4ddd	䷭	4ded	䷽	4dfd
䷎	4dce	䷞	4dde	䷮	4dee	䷾	4dfe
䷏	4dcf	䷟	4ddf	䷯	4def	䷿	4dff

All the above hexagrams are typeset with Symbola. Other options, particularly for embedding in the case of the first three and those above, include Noto Sans Symbols, Droid Sans Fallback, Free Serif, Cormorant Garamond, Times New Roman, Arial Unicode MS, Apple Symbols and even Garamond Premier Pro.

Symbola:	䷀ ䷁	Times New Roman:	䷀ ䷁
Noto Sans Symbols:	䷀ ䷁	Droid Sans Fallback:	䷀ ䷁
Free Serif:	䷀ ䷁	Free Sans:	䷀ ䷁
Cormoramt Garamond:	䷀ ䷁	Cormoramt Infant:	䷀ ䷁
Arial Unicode MS:	䷀ ䷁	Apple Symbols:	䷀ ䷁
Garamond Premier Pro:	䷀ ䷁	Code2000:	䷀ ䷁

易經	*I Ching* (Traditional)	易经	*I Ching* (Simplified)
大班	Taipan (Tai-Pan)[319]	麻[320]	hemp[321] (Traditional)

English vs. (mostly) Simplified Chinese Typeface Differences

大班	Verdana	大班	Arial
大班	Helvetica	大班	Arial Unicode MS
大班	Symbola	大班	Times New Roman
大班	Code2000	大班	Droid Sans Fallback
大班	Garamond Premier Pro	大班	Times
大班	Adobe Garamond Pro	大班	Adobe Jenson Pro
大班	Adobe Caslon Pro	大班	Adobe Devangari
大班	**Adobe Fan Heiti Std**	大班	Adobe Fangsong Std
大班	Adobe Arabic	大班	Adobe Hebrew
大班	Cormorant Garamond	大班	Cormorant
大班	Cormorant Infant	大班	CORMORANT SC
大班	CORMORANT UNICASE	大班	Cormorant Upright
大班	Everson Mono	大班	Apple Symbols
大班	Free Serif	大班	Apple SD Gothic Neo
大班	Free Sans	大班	Roboto
大班	Free Mono	大班	SourceCodePro
大班	Source Sans Pro	大班	Source Serif Pro
大班	STIX General	大班	Tahoma
大班	Monaco	大班	Menlo
大班	Merge One	大班	Merriweather Sans
大班	Minion Pro	大班	Merriweather

319 In general *taipan* refers to the Australasian snake, especially the aggressive Coastal Taipan and the more lethal, but not as aggressive, Inland Taipan (AKA the Fierce Snake, so named for its venom rather than its attitude). Whereas *tai-pan* generally refers to the Chinese title often attributed to leaders of certain types of organisations (e.g. the James Clavell novel of the same name).
320 https://en.wikipedia.org/wiki/Radical_200
321 https://en.wikipedia.org/wiki/Má

English vs. (mostly) Simplified Chinese Typeface Differences

大班	Fira Sans	大班	Fira Mono
大班	Gentium Basic	大班	Gentium Book Basic
大班	Gentium Plus	大班	Nimbus Roman No. 9
大班	Liberation Serif	大班	Liberation Mono
大班	Liberation Sans	大班	Liberation Sans Narrow
大班	Linux Biolinum G.	大班	Linux Biolinum O.
大班	Linux Libertine G.	大班	Linux Libertine Display G.
大班	Linux Libertine O.	大班	Linux Libertine Display O.
大班	Tinos	大班	Arimo
大班	Noto Serif	大班	Noto Sans Symbols
大班	Noto Sans	大班	Noto Mono
大班	Noto Sans CJK SC	大班	Noto Sans Mono CJK SC
大班	Noto Sans CJK TC[322]	大班	Noto Sans Mono CJK TC
大班	Athelas[323]	大班	Charter[324]
大班	Georgia[325]	大班	Iowan Old Style[326]
大班	Palatino[327]	大班	Seravek[328]

Note: For more comprehensive examples which indicate both font cohesiveness and the specific Unicode glyph coverage per text, refer to the UnicodeText document in this directory.[329] Some fonts may or may not include examples of different faces within that family (e.g. underline, *italics* and **bold** characters, as well as things like small capitals, with *or* **without** **prior** ***variations***).

322 As the font name indicates, this is Traditional Chinese for the initial characters and not simplified Chinese. The same applies to the Noto Sans Mono CJK TC font on the same line.
323 Bundled with iBooks. May be common amongst Apple eReaders.
324 Bundled with iBooks. May be common amongst Apple eReaders.
325 Bundled with iBooks and Kobo software. May be common amongst Apple eReaders, possibly all eReaders.
326 Bundled with iBooks. May be common amongst Apple eReaders.
327 Bundled with iBooks. May be common amongst Apple eReaders.
328 Bundled with iBooks. May be common amongst Apple eReaders.
329 Not currently distributed as the PDF version of this document is due to font embedding and licensing restrictions.

Roman Numerals

Though there is Unicode for Roman numerals from U+2160–217F, it is generally best to use the characters in an existing font. Thus it is better to maintain a useful key for those characters here.

Note that the over lining uses the character settings and the side lines are pipes.

Base Characters

I	1	V	5	X	10	L	50
C	100	D	500	M	1,000		
|I|	100	|V|	500	|X|	1,000	|L|	5,000
|C|	10,000	|D|	50,000	|M|	100,000		
$\overline{\text{I}}$	1,000	$\overline{\text{V}}$	5,000	$\overline{\text{X}}$	10,000	$\overline{\text{L}}$	50,000
$\overline{\text{C}}$	100,000	$\overline{\text{D}}$	500,000	$\overline{\text{M}}$	1,000,000		
$\overline{\text{\|I\|}}$	100,000	$\overline{\text{\|V\|}}$	500,000	$\overline{\text{\|X\|}}$	1,000,000	$\overline{\text{\|L\|}}$	5,000,000
$\overline{\text{\|C\|}}$	10,000,000	$\overline{\text{\|D\|}}$	50,000,000	$\overline{\text{\|M\|}}$	100,000,000		

Preferred Characters

I	1	V	5	X	10	L	50
C	100	D	500	M	1,000	$\overline{\text{V}}$	5,000
$\overline{\text{X}}$	10,000	$\overline{\text{L}}$	50,000	$\overline{\text{C}}$	100,000	$\overline{\text{D}}$	500,000
$\overline{\text{M}}$	1,000,000	$\overline{\text{\|L\|}}$	5,000,000	$\overline{\text{\|C\|}}$	10,000,000	$\overline{\text{\|D\|}}$	50,000,000
		$\overline{\text{\|M\|}}$	100,000,000				

Official Unicode Characters

I	2160	II	2161	III	2162	IV	2163
V	2164	VI	2165	VII	2166	VIII	2167
IX	2168	X	2169	XI	216a	XII	216b
L	216c	C	216d	D	216e	M	216f

i	2170	ii	2171	iii	2172	iv	2173
v	2174	vi	2175	vii	2176	viii	2177
ix	2178	x	2179	xi	217a	xii	217b
l	217c	c	217d	d	217e	m	217f

Note that the overlining and sidelining with pipes will work with the official characters too, but there are no guarantees that every font will have the characters included. By sticking with the regular characters in the latin alphabet, this potential problem is avoided.

Official Unicode Characters Overlined

$\overline{\mathrm{I}}$	2160	$\overline{\mathrm{II}}$	2161	$\overline{\mathrm{III}}$	2162	$\overline{\mathrm{IV}}$	2163
$\overline{\mathrm{V}}$	2164	$\overline{\mathrm{VI}}$	2165	$\overline{\mathrm{VII}}$	2166	$\overline{\mathrm{VIII}}$	2167
$\overline{\mathrm{IX}}$	2168	$\overline{\mathrm{X}}$	2169	$\overline{\mathrm{XI}}$	216a	$\overline{\mathrm{XII}}$	216b
$\overline{\mathrm{L}}$	216c	$\overline{\mathrm{C}}$	216d	$\overline{\mathrm{D}}$	216e	$\overline{\mathrm{M}}$	216f

$\overline{\mathrm{i}}$	2170	$\overline{\mathrm{ii}}$	2171	$\overline{\mathrm{iii}}$	2172	$\overline{\mathrm{iv}}$	2173
$\overline{\mathrm{v}}$	2174	$\overline{\mathrm{vi}}$	2175	$\overline{\mathrm{vii}}$	2176	$\overline{\mathrm{viii}}$	2177
$\overline{\mathrm{ix}}$	2178	$\overline{\mathrm{x}}$	2179	$\overline{\mathrm{xi}}$	217a	$\overline{\mathrm{xii}}$	217b
$\overline{\mathrm{l}}$	217c	$\overline{\mathrm{c}}$	217d	$\overline{\mathrm{d}}$	217e	$\overline{\mathrm{m}}$	217f

Mac OS X Alternate Characters

This is a list of characters inserted by pressing Alt+[key] and Alt+Shift+[key]. Leaving the ` character out (the same key as tilde) because it only ever produces the same character.

Alt+[key] characters

1	¡	2	™	3	£	4	¢	5	∞
6	§	7	¶	8	•	9	ª	0	º
-	–	=	≠	[	“	]	‘	\	«
'	æ	;	…	,	≤	.	≥	/	÷
a	å	b	∫	c	ç	d	∂	e	´
f	ƒ	g	©	h	˙	i	ˆ	j	∆
k	˚	l	¬	m	µ	n	˜	o	ø
p	π	q	œ	r	®	s	ß	t	†
u	¨	v	√	w	∑	x	≈	y	¥
z	Ω								

Alt+Shift+[key] characters

1	⁄	2	€	3	‹	4	›	5	ﬁ
6	ﬂ	7	‡	8	°	9	·	0	‚
-	—	=	±	[	”	]	’	\	»
'	Æ	;	Ú	,	¯	.	˘	/	¿
a	Å	b	ı	c	Ç	d	Î	e	´
f	Ï	g	˝	h	Ó	i	ˆ	j	Ô
k		l	Ò	m	Â	n	˜	o	Ø
p	∏	q	Œ	r	‰	s	Í	t	ˇ
u	¨	v	◊	w	„	x	˛	y	Á
z	¸								

Emacs Init File

The following is the parts of my `$HOME/.emacs` file which affect fonts and glyph representation as described above. Note that the relatively high font point size is due to sitting a reasonable distance from a high resolution screen and wishing to read comfortably.

It should be possible to copy and paste in place, but it can also be provided in a gist if necessary.

```
(global-set-key (kbd "<f8>") 'insert-char)
(global-set-key (kbd "M-<f8>") 'describe-char)
(global-set-key (kbd "<f13>") 'describe-char)
;;
;; Specify fonts
;;
;; set a default font
;;
(when (member "Inconsolata" (font-family-list))
  (set-face-attribute 'default nil :font "Inconsolata Medium 24"))
;;
;; with default fallback support (the last one takes priority)
;; Specific code point range overrides by fallback
;;
;;
;; Plane 0
;; Basic Multilingual Plane (BMP) – 0x0000 to 0xffff.
;;
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 20 :name "Arial Unicode MS"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 24 :name "FreeSerif"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 23 :name "FreeSans"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 22 :name "FreeMono"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 24 :name "Droid Sans Symbols"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 24 :name "Symbola"))
(set-fontset-font "fontset-default" '(#x0000 . #xffff)
        (font-spec :size 24 :name "Menlo"))
;;
(set-fontset-font "fontset-default" '(#x0000 . #x25ff)
        (font-spec :size 20 :name "Source Code Pro Medium"))
(set-fontset-font "fontset-default" '(#x0000 . #x1ef9)
        (font-spec :size 24 :name "Inconsolata"))
(set-fontset-font "fontset-default" '(#x0300 . #x03ff)
        (font-spec :size 20 :name "Source Code Pro Medium"))
(set-fontset-font "fontset-default" '(#x0400 . #x04ff)
        (font-spec :size 20 :name "Source Code Pro Medium"))
(set-fontset-font "fontset-default" '(#x1efa . #x25ff)
        (font-spec :size 20 :name "Source Code Pro Medium"))
(set-fontset-font "fontset-default" '(#x2600 . #x26ff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2700 . #x27ff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2800 . #x28ff)
        (font-spec :size 24 :name "FreeMono"))
(set-fontset-font "fontset-default" '(#x2900 . #x29ff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2a00 . #x2aff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2b00 . #x2bff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2c00 . #x2cff)
        (font-spec :size 20 :name "Andika"))
(set-fontset-font "fontset-default" '(#x2d00 . #x2dff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2e00 . #x2e7f)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x2e80 . #x2eff)
        (font-spec :size 20 :name "Arial Unicode MS"))
(set-fontset-font "fontset-default" '(#x2f00 . #x2fff)
        (font-spec :size 24 :name "Noto Sans CJK TC"))
(set-fontset-font "fontset-default" '(#x3000 . #x3fff)
        (font-spec :size 20 :name "Arial Unicode MS"))
(set-fontset-font "fontset-default" '(#x4000 . #x4dbf)
        (font-spec :size 20 :name "Arial Unicode MS"))
(set-fontset-font "fontset-default" '(#x4dc0 . #x4dff)
        (font-spec :size 24 :name "Symbola"))
(set-fontset-font "fontset-default" '(#x4e00 . #x9fea)
        (font-spec :size 20 :name "Arial Unicode MS"))
(set-fontset-font "fontset-default" '(#x9feb . #xdfff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
;;
;; 0xe000 to 0xf8ff is reserved private use and unlikely to be
;; actively used.  The same is true of 0xf0000 to 0x10ffff.
;;
(set-fontset-font "fontset-default" '(#xe000 . #xf8ff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
;;
(set-fontset-font "fontset-default" '(#xf900 . #xffff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
;;
;; Plane 1
;; Supplementary Multilingual Plane (SMP) - 0x10000 to 0x1ffff.
;;
(set-fontset-font "fontset-default" '(#x10000 . #x101ff)
        (font-spec :size 24 :name "Code2001"))
(set-fontset-font "fontset-default" '(#x10100 . #x101fd)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x10300 . #x104ff)
        (font-spec :size 24 :name "Code2001"))
(set-fontset-font "fontset-default" '(#x10800 . #x1091f)
        (font-spec :size 24 :name "Code2001"))
(set-fontset-font "fontset-default" '(#x12000 . #x1254f)
        (font-spec :size 24 :name "Noto Sans Cuneiform"))
```

```
(set-fontset-font "fontset-default" '(#x13000 . #x1343f)
        (font-spec :size 24 :name "Noto Sans Egyptian Hieroglyphs"))
(set-fontset-font "fontset-default" '(#x16800 . #x16a3f)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x1d000 . #x1d7ff)
        (font-spec :size 24 :name "FreeSerif"))
(set-fontset-font "fontset-default" '(#x1d800 . #x1f1e5)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f1e6 . #x1f1ff)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x1f200 . #x1f2ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f300 . #x1f3ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f400 . #x1f4ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f500 . #x1f5ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f600 . #x1f6ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f700 . #x1f773)
        (font-spec :size 24 :name "Noto Sans Symbols"))
(set-fontset-font "fontset-default" '(#x1f774 . #x1f7ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f800 . #x1f8ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1f900 . #x1f9ff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1fa00 . #x1faff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1fb00 . #x1fbff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1fc00 . #x1fcff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1fd00 . #x1fdff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1fe00 . #x1feff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
(set-fontset-font "fontset-default" '(#x1ff00 . #x1ffff)
        (font-spec :size 24 :name "Twitter Color Emoji"))
;;
;; Plane 2
;; Supplementary Ideographic Plane (SIP) - 0x20000 to 0x2ffff.
;;
(set-fontset-font "fontset-default" '(#x20000 . #x2ff7f)
        (font-spec :size 24 :name "Noto Sans Mono CJK TC"))
(set-fontset-font "fontset-default" '(#x2ff80 . #x2ffff)
        (font-spec :size 24 :name "Noto Sans Mono CJK SC"))
;;
;; Planes 3 to 13
;;
(set-fontset-font "fontset-default" '(#x30000 . #xdffff)
        (font-spec :size 24 :name "Unifont Upper CSUR"))
(set-fontset-font "fontset-default" '(#x30000 . #xdffff)
        (font-spec :size 24 :name "Unifont Upper"))
;;
;; Plane 14
;; Supplementary Special Plane (SSP) - 0xe0000 to 0xeffff.
;;
(set-fontset-font "fontset-default" '(#xe0000 . #xeffff)
        (font-spec :size 24 :name "Unifont Upper CSUR"))
(set-fontset-font "fontset-default" '(#xe0000 . #xeffff)
        (font-spec :size 24 :name "Unifont Upper"))
;;
;; The next sections cover planes 15 and 16 which are reserved for
;; private use and are unlikely to be covered by any publicly
;; available font.
;;
;; Plane 15
;;
(set-fontset-font "fontset-default" '(#xf0000 . #xfffff)
        (font-spec :size 24 :name "Unifont Upper CSUR"))
(set-fontset-font "fontset-default" '(#xf0000 . #xfffff)
        (font-spec :size 24 :name "Unifont Upper"))
;;
;; Plane 16
;;
(set-fontset-font "fontset-default" '(#x100000 . #x10ffff)
        (font-spec :size 24 :name "Unifont Upper CSUR"))
(set-fontset-font "fontset-default" '(#x100000 . #x10ffff)
        (font-spec :size 24 :name "Unifont Upper"))
;;
;; End of font fallback settings
```