

Dubberly Design Office

Understanding Digital Typography

2

Version 1.8
June 22, 2011

Dubberly Design Office
2501 Harrison Street, #7
San Francisco, CA 94110

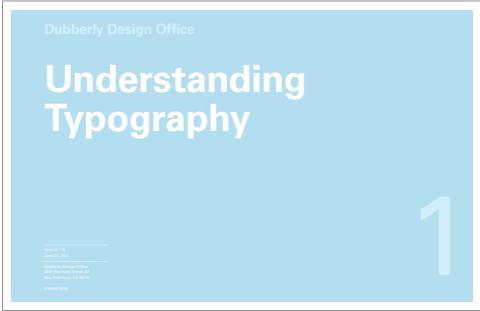
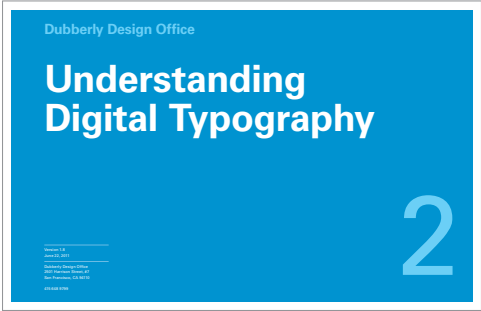
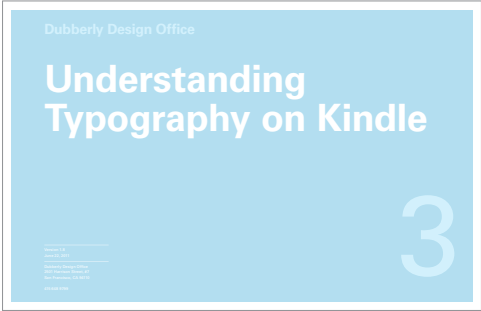

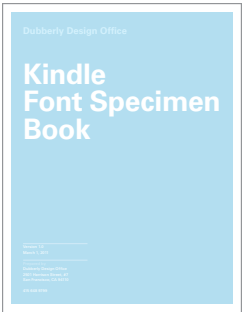
415 648 9799

Overview

Dubberly Design Office analyzed typography on Kindle and suggested ways the experience of reading might be improved. We have produced a sequence of four reports leading to specific recommendations.

The first report provides an overview of typography; the second describes how computers have changed type and typesetting; and the third describes how Kindle implements type and typography today. We also produced a Kindle Font Specimen Book as a supplement to the third report. The fourth report suggests ways the experience of reading on Kindle might be improved.

All four reports are organized in a similar structure, beginning with an overview and preceding from glyphs to pages to books to collections for Latin and other character sets.

				
	How Are Languages Represented?	What's Different?	What is Kindle?	
Glyphs	How Have Latin Characters Evolved? How Have Fonts Evolved? How Do You Make Letters Look Good?	How Is Type Input? How Is Type Encoded? How Are Characters Represented? How Do Computers Display Type? How Are Fonts Managed?	How Does Kindle Encode Fonts? How Does Kindle Render Fonts? * Kindle Font Specimen Book	Fonts
Pages	How Do Characters Go Together? How Do Words Go Together?	How Is Text Formatted? How Are Pages Rendered?	What Can Users Change When Reading a Kindle Book?	Layout
Books	How Do Pages Go Together?	What Is a Digital Book?	How Does Kindle Encode Documents? How Does Kindle Render Documents?	Substrate
Collections	What Is a Book?	How Are Digital Books Managed?	How Do You Publish a Kindle Book?	Interactivity
		How Do Fonts Work on the Web?	<div><div>*</div></div>	The Future

Contents:

Understanding Digital Typography

4	Introduction	55	How Are Characters Represented?	82	How Is Text Formatted?	105	What Is a Digital Book?
5	Timeline of Publishing Approaches	56	Bitmap Fonts	83	Markup & Style	106	Access
6	What's Different?	57	Digital Font File Formats (Timeline)	84	Plain Text	107	Hypertext
7	Analog vs Digital	58	Ikarus	85	Evolution of the Typesetting "Stack"	108	Memex
8	Screen Technology	59	Metafont	86	TeX	109	E-book Formats
9	Electronic Ink	60	PostScript & TrueType	87	PDF		
10	Stack Components	61	OpenType	88	HTML	110	How Are Digital Books Managed?
11	File Formats, Languages, Libraries	62	Contemporary Font Format Comparison	89	CSS	111	Directories
12	How Is Type Input?	63	How Do Computers Display Type?	90	JavaScript	112	Tagging
13	Mechanical Layout	64	Bit Depth	91	DOM	113	Search
14	Visual & Functional Layout	65	Outlines vs Pixels	92	Render Tree	114	Management Tools
15	Virtual Keyboard	66	Hinting	93	CMS	115	Online Management Tools
16	QWERTY	67	Aliasing	94	XML	116	Online Social Book Services
17	Dvorak Simplified	68	Anti-aliasing	95	Dublin Core	117	Types of E-book Readers
18	Key Strokes to Words	69	Subpixel Rendering	96	How Are Pages Rendered?		Appendix:
19	Language Support	70	Comparing Rendering Strategies	97	Hello, World!	118	How Do Fonts Work on the Web?
20	Input Method Editors	71	Rendering Engines Comparison	98	Font Cache	119	Browser Layout Engines
21	How Is Type Encoded?	72	Windows vs Mac	99	Glyph Cache	120	Webpage Rendering
22	Counting for Computers	73	OS vs Browser	100	Contextual Shaping	121	Webpage Rendering: Fonts
23	Morse Code	74	How Are Fonts Managed?	101	Line & Paragraph Assembly	122	Font Replacement
24	Baudot Code	75	System Fonts	102	H&J	123	Font Hosting
25	Murray Code	76	Web-safe Fonts	103	Page Assembly	124	EOT
26	ASCII	77	Font Management Software	104	Frame Buffer	125	WOFF
27	Code-pages	78	Font Foundries			126	Digital Rights Management for Fonts
28	Windows 1252	79	Font Embedding			127	SVG
29–45	ISO/IEC 8859	80	Font Substitution			128	Font Hosting: Google
46	Big5	81	Font Linking			129	Font Hosting: Typekit
47	Shift JIS						
48	KS X 1001						
49–53	Unicode						
54	GB18030						

Introduction

Typography is intimately intertwined with technology. As new marking technologies arrive, they reproduce earlier forms, but shortly thereafter they begin to modify those forms and adapt them to the grain of the new technology. The shape of serif letters, for example, is a reflection of the tools with which they are drawn or carved – serifs are a neat way of ending a stroke drawn by a brush.

Book making faced a major disruption in 1456, when Gutenberg introduced printing with moveable type in the West. (It was also invented separately in Asia.) Book making changed again in the 1880s with the introduction of mechanical typesetting equipment. The process of change accelerated in the 1960s, first with phototypesetting and then with digital typesetting.

The scale of change in the past 50 years has been immense. No aspect of books, book making, and book selling has been left untouched. What it means “to read” and what constitutes “a book” is changing rapidly.

This report describes issues unique to digital type and typography: how digital text is input and encoded; how letterforms are represented, rendered, and managed; and how text is formatted and rendered. This report also touches on digital books and digital libraries since they cannot be entirely separated from digital typography.

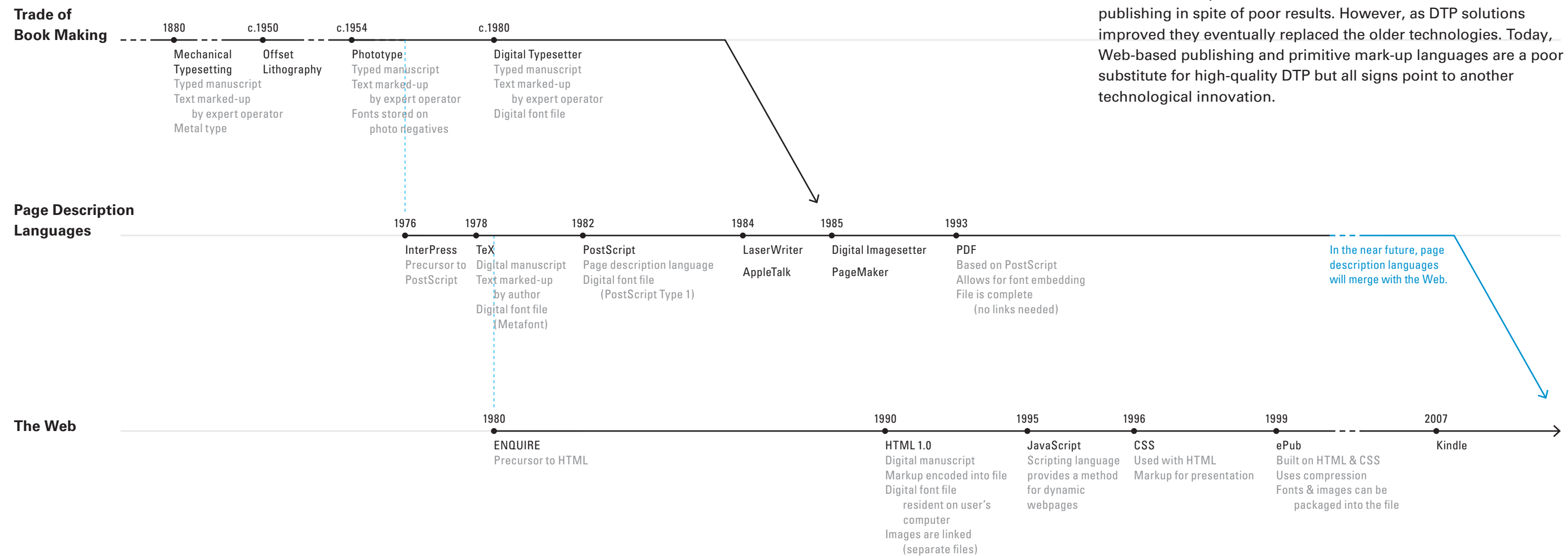
Technologies related to type and typography on the Web are rapidly evolving. The main issues are covered in the appendix.

Timeline of Publishing Approaches

Typesetting evolved rapidly in the 1970s and 1980s. In parallel, low-end desktop computer systems became increasingly sophisticated. By the mid-1990s, typesetting as an independent service began to disappear. At first, it was replaced by service bureaus offering image-setting at substantially lower prices than traditional typesetting businesses. Service bureaus didn't last long as commercial printers began to offer direct-to-plate technology, cutting out the need for traditional typesetting, paste-up, camera work, stripping, and plate making. At that point, traditional book making essentially merged with desktop design.

For now, plate-making is the final step in the book production process, but the days of offset-lithography printing are limited. Digitally-driven ink-jet printing will replace offset-lithography for most applications within the next ten years.

The low-cost and wide availability of early desktop publishing (DTP) solutions provided a viable alternative to traditional publishing in spite of poor results. However, as DTP solutions improved they eventually replaced the older technologies. Today, Web-based publishing and primitive mark-up languages are a poor substitute for high-quality DTP but all signs point to another technological innovation.



What's Different?

In the last 50 years, technology has evolved at a rapid pace, which has had a profound impact on all aspects of typography.

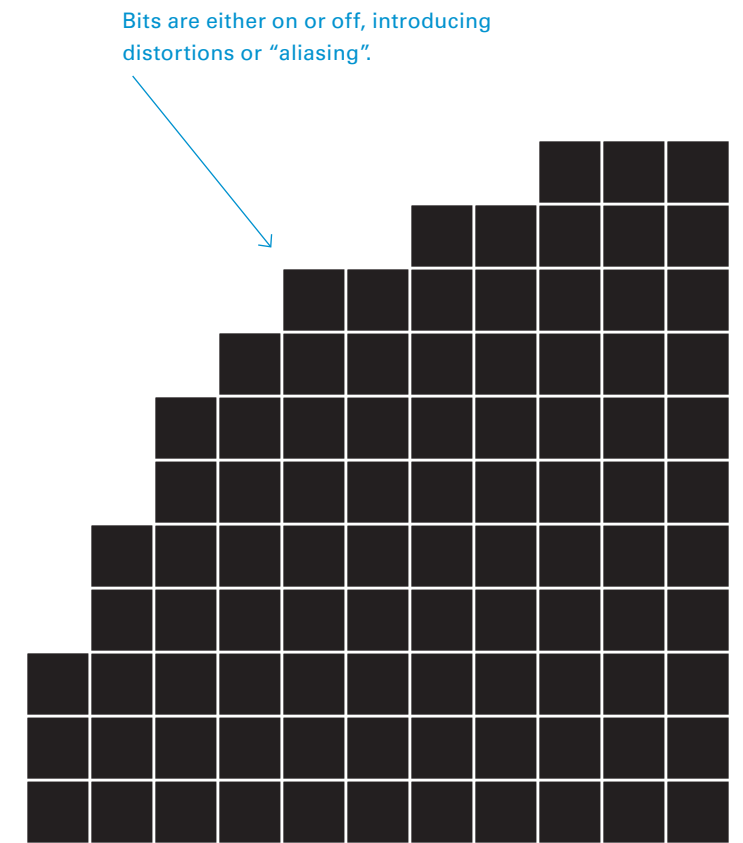
What's Different?

Analog vs Digital

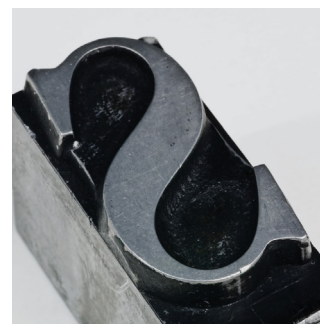
Prior to digital technology, letters were written with a brush or pen, carved from metal, painted on film, or cut from rubylith. All of these processes are analog (i.e. physically continuous). This worked well because letterforms can have complex shapes and most were printed very small. Digital technology changed everything because it breaks up information into discrete chunks (i.e. ones and zeros).



Analog curve



Digital curve



Metal type



Stone carving



Tile signage



Needlepoint embroidery

Digital type predates computers.

What's Different?

Screen Technology

Letters are now rendered on screens in addition to being printed on paper. There are many different types of screens.



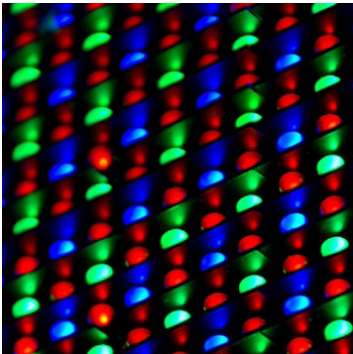
Segment Display

The screen is comprised of preconfigured segments, which switch on or off to create desired glyphs. The segments are usually light emitting diodes (LED) or liquid crystal displays (LCD), though they may also use vacuum fluorescent, cold cathode gas discharge, incandescent filaments, and other technologies to emit light. Segments may also be purely mechanical and rely on reflected ambient light to be legible.



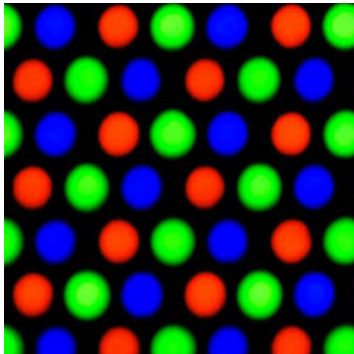
Dot Matrix

The screen is a matrix of lights or mechanical indicators arranged in a rectangular configuration (other configurations are possible, but are not common). Text and graphics can be rendered by switching indicators on or off.



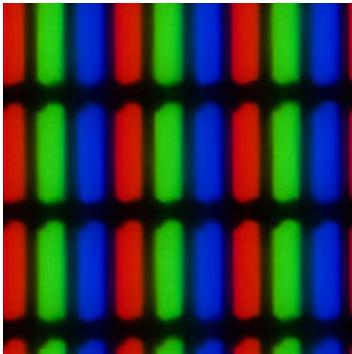
Light-Emitting Diode (LED)

Like an incandescent light bulb, an LED passes an electrical signal up one electrical terminal, across a gap, and down another. Unlike an incandescent bulb, LEDs do not have a filament, instead they have a diode – a kind of simple semiconductor with two halves. Only diodes made of certain materials light up, the material determines the color of light.



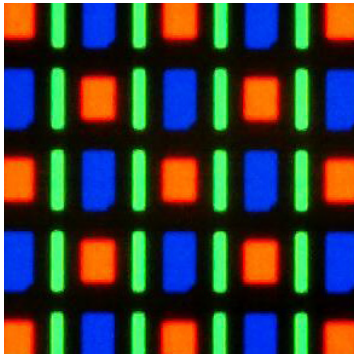
Cathode Ray Tube (CRT)

CRTs use an electron beam (the “ray” in “cathode ray tube”) to “paint” on a screen of phosphors (a material that emits visible light when struck with radiation). The beam paints one line at a time, from left to right, then moves to the line below it, and repeats this process until it reaches the bottom of the screen. In a black and white CRT, the phosphor is a solid coat of white. In a color CRT, there are phosphors which emit red, green, and blue light, packed together in clusters called “triads”.



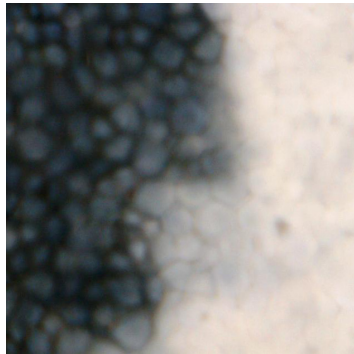
Liquid Crystal Display (LCD)

In color LCDs each individual pixel is divided into three cells, or subpixels, which are colored red, green, and blue, respectively. Color is achieved by passing light through filters of pigment, dyes, or metal oxides.



Organic Light-Emitting Diode (OLED)

Unlike most display technology, OLEDs do not use a backlight to illuminate pixels. Instead, they are composed of an organic layer sandwiched between two layers of electrodes. The organic molecules emit light. When they conduct electricity between the two electrode layers.



Electronic Ink (eInk)

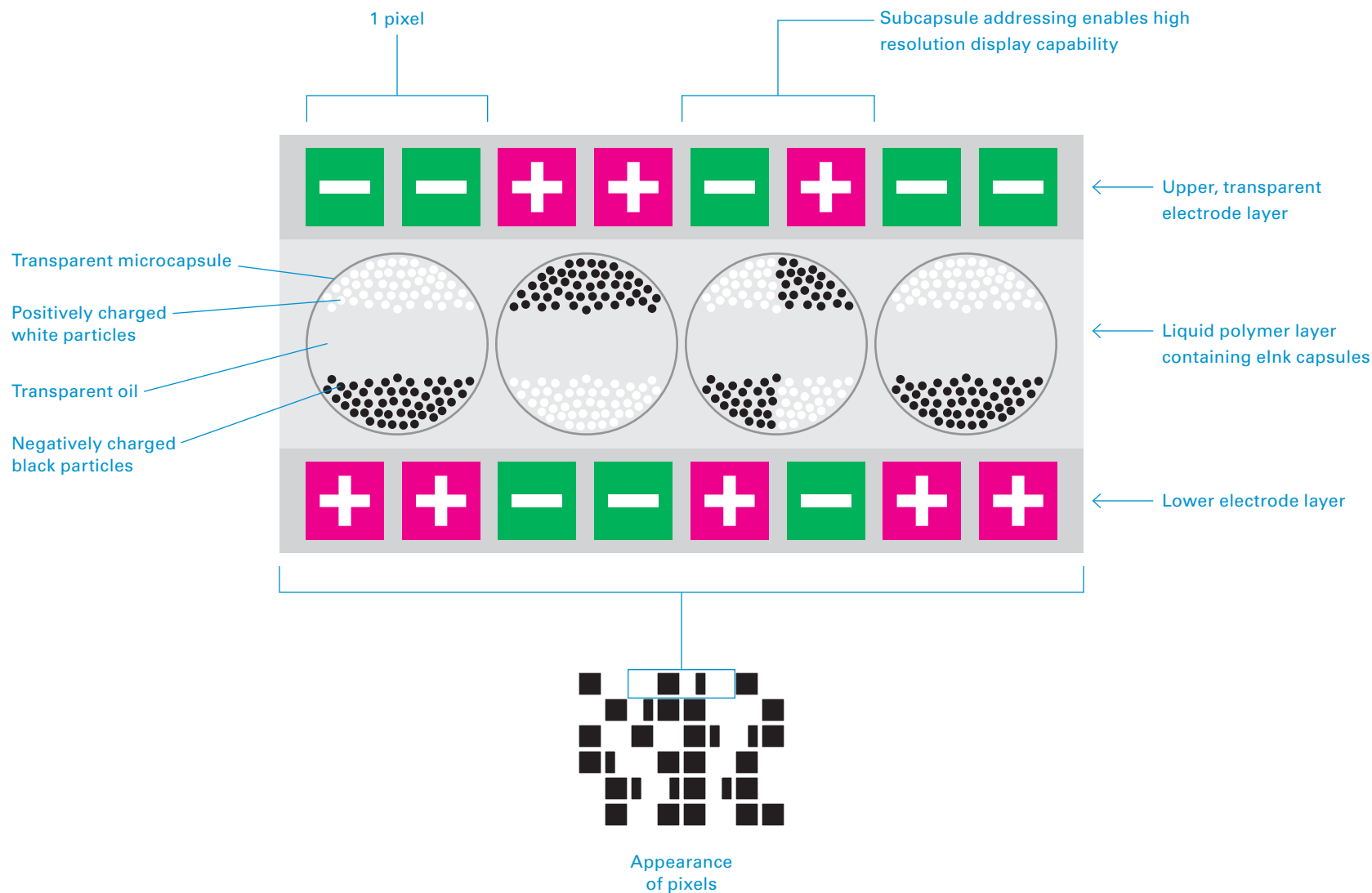
eInk uses microcapsules held in a thin layer of liquid polymer sandwiched between two arrays of electrodes. The microcapsules contain numerous white and black pigment particles suspended in a clear solution. The white particles are negatively charged – applying a positive charge to the top of the micro-capsule pulls the white particles to the top, and the capsule appears white. The external charge is reversed to turn the capsule black. Like OLED technology, eInk also uses no backlight, but it requires ambient light to be legible.

What's Different?

Electronic Ink

Unlike conventional backlit flat panel displays, electronic ink (eInk) displays have no backlight and thus reflect light like ordinary paper. The technology is a subset of “bistable” display technology, which retains an image when powered off (bistable because the image displayed is stable in two states: on and off).

The main principle in eInk is the use of tiny microcapsules (or, “bubbles”) that contain both black and white pigment particles suspended in a clear fluid. Each microcapsule is the size of one pixel and they are sandwiched between two layers of electrodes (the top of which is clear). When electricity is passed between electrode pairs, it causes the particles in each microcapsule to either rise or sink. A positive charge causes white particles to rise while a negative charge causes black particles to rise. It is possible to generate gray-tones by applying a partial charge to a microcapsule – current eInk technology can achieve 16 gray-tones (or, 4-bit color depth).

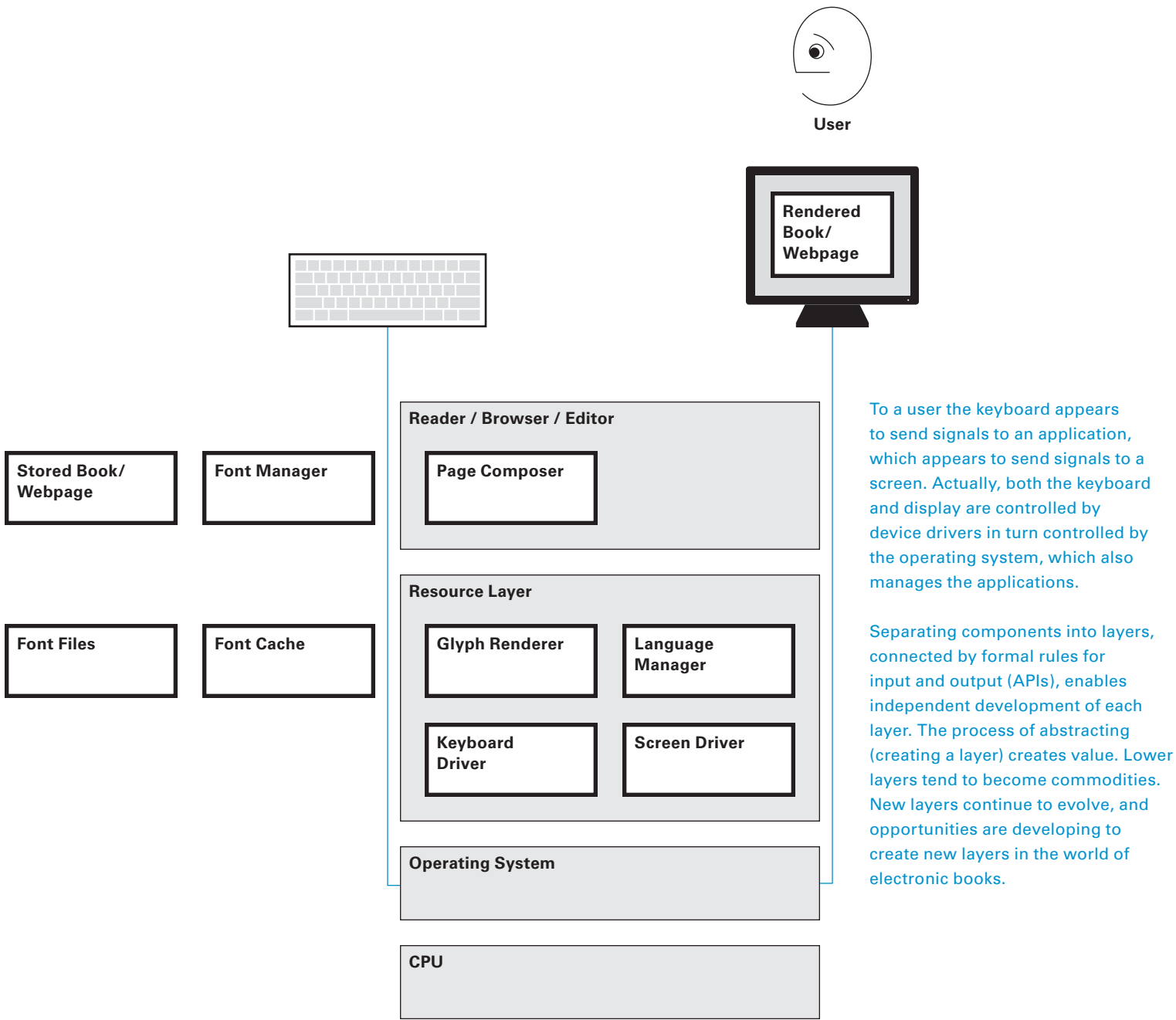


Disclaimer:
It is unclear how exactly grayscale is produced with eInk – no thorough explanation could be found and the literature is vague.

What's Different?

Stack Components

Computer technology can typically be described in terms of a “stack”. A stack is a set of components that build on top of one another, each layer utilizing the layer below it for core components and enabling the layer above to function in the same way. For instance, a page composer relies on the elements of the resource layer such as the glyph renderer and keyboard driver to function. These resources in turn need the OS to function properly, and the OS relies on the CPU for computing power. This process also works in the other direction: without an OS, the CPU would have nothing to do.



What’s Different?

File Formats,

File Formats

A file format is method of encoding information into a form that a computer can understand (0s and 1s, i.e. binary).

e.g. PostScript page description file format (.ps)
 JPEG image file format (.jpeg)
 MPEG-2 Audio Layer III file format (.mp3)

Font File Formats

A format for encoding the glyph outlines, metrics, hinting data, context-specific features, etc. of a font.

e.g. PostScript Type 1 (.pfm)
 TrueType (.ttf)
 OpenType (.otf)
 Web Open Font Format (.woff)

Image File Formats

Standardized means of organizing and storing digital images.

Image file formats store data as either pixels or vector data that are rasterized to pixels when displayed on screen.

e.g. JPEG (.jpg)
 TIFF (.tif)
 Encapsulated PostScript (.eps)

Electronic Publishing Formats

A format for packaging text and image-based publications in digital form. Many make use of compression algorithms as a way to contain file size and package more than one file (text, HTML, image, fonts) into a single transportable file.

e.g. Plain Text (.txt)
 ePub (.epub)
 Mobipocket (.mobi, .prc)
 Portable Document Format (.pdf)

Languages,

Programming Language

An artificial language designed for writing instructions that can be performed by a computer or other machinery. The earliest programming languages predate the invention of the computer and were used for controlling looms in textile manufacturing.

e.g. C++
 Java
 Ruby

Markup Language

A system for annotating text in a way that is syntactically distinguishable from that text. Markup languages are typically used as a way to instruct a person or computer about what the various parts of a text are.

e.g. HTML (HyperText Markup Language)
 TeX
 XML (Extensible Markup Language)

Style Sheet Language

A language for expressing the presentation of a document.

e.g. CSS (Cascading Style Sheets)
 XSL (Extensible Style Sheet Language)

Scripting Language

A programming language that allows control of an application.

“Scripts” are distinct from the core code and usually written in a different language. Scripting languages typically allow a user to modify the default arrangement of an application.

e.g. JavaScript
 PHP
 ActionScript

Page Description Language

Used to describe the appearance of a printed page at a higher level than an output bitmap. Most page description languages are not full programming languages but contain selective elements of them.

e.g. PostScript
 SVG (Scalable Vector Graphics)
 InterPress

Libraries

Software Library

A collection of resources used to develop software, including subroutines, classes, values, and type specifications.

e.g. Java Class Library
 C++ Standard Library
 Dynamic Link Library (DLL)

Application Framework

An abstraction in which common code providing generic functionality can be quickly called or selectively overridden by user code to provide more tailored functionality.

e.g. Ruby on Rails (Web framework for Ruby)
 Django (Web framework for Python)
 Agavi (application framework for PHP)

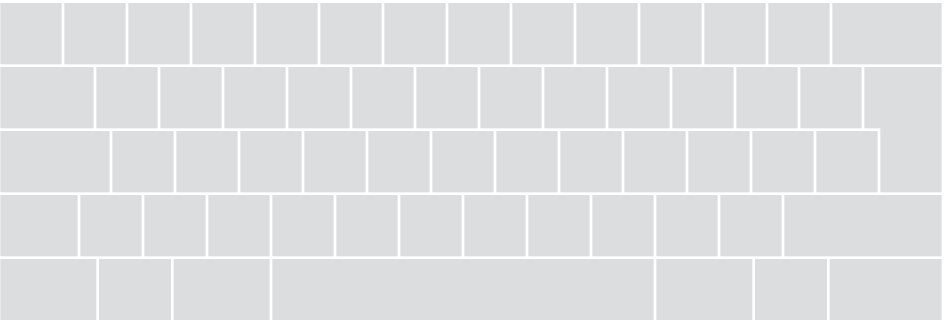
How Is Text Input?

Writing systems are relatively simple to apply to paper with a pen, pencil, or brush. However entering letters and words into a computer is a little more complicated. Some writing systems, such as English, are fairly straightforward. The user enters one character after another and separates words with spaces. There is one key for each letter, and a modifier key (Shift) to alternate between lower- and upper-case. However, many writing systems have far more characters than English, and input is not nearly as linear. A simple example is Norwegian, which also uses a Latin-based script, but requires the use of many more diacritical marks, resulting in the user's Shift, Option, and Command keys getting a workout. A more complex example is Japanese, which has three different scripts and a variety of ways to enter them. A truly international text display and management system needs to take these and many other writing systems into account.

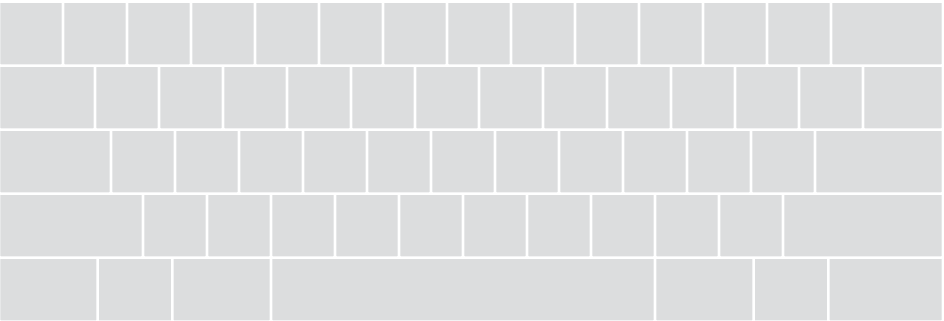
How Is Type Input?

Mechanical Layout

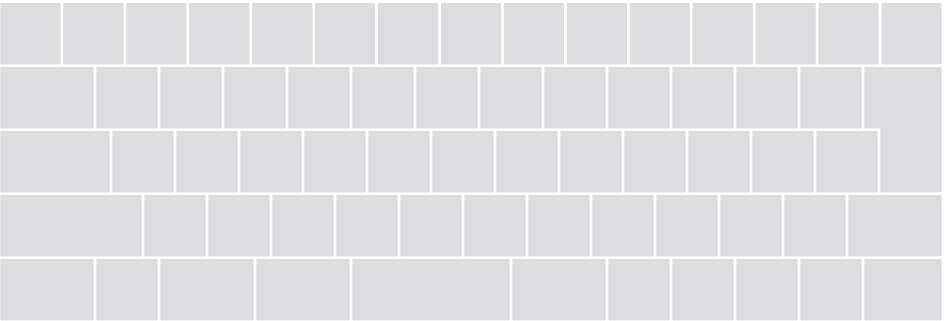
A keyboard sends key codes to the computer. Software, usually part of the operating system, determines how the key codes are interpreted. (This interpretation method can typically be changed by the user). Most keyboards today use one of three different mechanical layouts, usually referred to as ISO (International Standards Organization), ANSI (American National Standards Institute), and JIS (Japanese Industrial Standards). **Mechanical layout refers only to the physical arrangement of keys.** When the user presses the “A” key, the keyboard send an internal reference number or “raw keycode” that corresponds to the left-most main key in the home row, not “A”.



ISO keyboard layout: 61 keys



ANSI keyboard layout: 60 keys



JIS keyboard layout: 66 keys

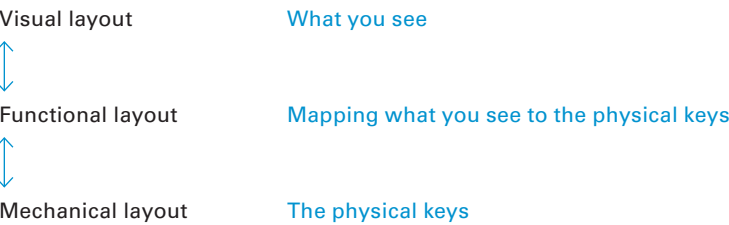
How Is Type Input?

Visual & Functional Layout

The visual layout of a keyboard is the set of marks and images displayed on each key. These tell the user what letter, figure, or mark will appear when a key is pressed. The same mechanical layout can be used with different visual layouts which vary by language, country, and user preference. For example, the ISO mechanical layout is used throughout Europe, but German, French, and UK variants have different visual layouts. (See page 47.)

The functional layout is the mapping of the mechanical layout to the visual layout. The functional layout is typically an operating system (OS) function and maps to the current OS language. The user can typically change the functional layout without changing the current OS language. This allows users to input any writing system, regardless of the keyboard’s visual layout. For example, a user may want to type in Hangul on an ANSI English keyboard.

Almost all keyboards have one to three more glyph sets than what’s shown on the visual layout. These additional glyphs are activated by pressing a modifier key in addition to a glyph key. With most Latin script system keyboards, the user must press the Shift key in addition to a character key to get an uppercase character. (It’s interesting to note that most Latin script system keyboards use uppercase characters in their visual layout—what you see on the keys is typically not what you get when you press them!) The control and Alt keys perform similar functions to input unmarked glyphs such as accented characters, mathematical characters, ligatures, or geometric symbols.



`	1	2	3	4	5	6	7	8	9	0	-	=	delete
tab	Q	W	E	R	T	Y	U	I	O	P	[]	enter
caps	A	S	D	F	G	H	J	K	L	;	'	#	
shift	\	Z	X	C	V	B	N	M	,	.	/		shift
cntrl	opt	cmd	space							cntrl	opt	cmd	

UK ISO keyboard layout

,	1	2	3	4	5	6	7	8	9	0	'	+	delete
tab	Q	W	E	R	T	Z	U	I	O	P	Š	Đ	enter
caps	A	S	D	F	G	H	J	K	L	Č	Ć	Ž	
shift	<	Z	X	C	V	B	N	M	,	.	-		shift
cntrl	opt	cmd	space							cntrl	opt	cmd	

Slovenian ISO keyboard layout

°	1	2	3	4	5	6	7	8	9	0	Ö	-	delete
tab	Q	W	E	R	T	Y	U	I	O	P	Ð	'	enter
caps	A	S	D	F	G	H	J	K	L	Æ	^	+	
shift	>	Z	X	C	V	B	N	M	,	.	Þ		shift
cntrl	opt	cmd	space							cntrl	opt	cmd	

Icelandic ISO keyboard layout

How Is Type Input?

Virtual Keyboard

A virtual keyboard is a software component that mimics the appearance and functionality of mechanical keyboards. Depending on hardware configuration, users “type” using a mouse-controlled pointer, by tapping directly on a touchscreen, or even with a mechanical keyboard. On a PC, virtual keyboards provide alternative input methods for the disabled, multi-lingual users who switch frequently between languages, and as a security measure to thwart keystroke logging malware. Devices that lack physical keyboards, such as smartphones and tablet computers, often use virtual keyboards as their sole text-input interface. Virtual keyboards often have the same visual layout as mechanical keyboards, although most stray from the standard ISO, ANSI, and JIS mechanical layout to save space and, especially on touchscreen devices, to maximize key size. This is often done by putting figures and punctuation marks on secondary screens.

One of the primary issues with virtual keyboards is how to provide feedback. This isn't a problem with mechanical keyboards because you can feel the key under your finger.



Motorola virtual keyboard.

How Is Type Input?

QWERTY

The most popular visual layout is known as QWERTY, **named after the first six letters on the keyboard**. QWERTY was invented in 1875 by Christopher Sholes and Amos Densmore. It is frequently claimed that QWERTY was developed by Sholes to slow typing speeds to prevent keys from jamming on early models of his typewriter. However, some researchers say this is an urban legend. Early target markets for typewriters were telegraphers who would transcribe incoming Morse code in real time – speed would have been essential to them. Furthermore, the three most common letter pairs in English – “th”, “er”, and “re” are all made by pressing adjacent keys.

Whatever the story behind QWERTY, it is undoubtedly an international standard. Variations on QWERTY are usually based on differences in the location of the “Q”, “A”, “Z”, “M”, and “Y” keys. As a result, Germans use QWERTZ keyboards and the French use AZERTY keyboards.

`	1	2	3	4	5	6	7	8	9	0	-	=	delete
tab	Q	W	E	R	T	Y	U	I	O	P	[]	\
caps	A	S	D	F	G	H	J	K	L	;	'	enter	
shift		Z	X	C	V	B	N	M	,	.	/	shift	
cntrl	opt	cmd	space							cntrl	opt	cmd	

United States QWERTY keyboard.

^	1	2	3	4	5	6	7	8	9	0	ß	'	delete
tab	Q	W	E	R	T	Z	U	I	O	P	Ü	+	enter
caps	A	S	D	F	G	H	J	K	L	Ö	Ä	#	
shift	<	Y	X	C	V	B	N	M	,	.	-	shift	
cntrl	opt	cmd	space								cntrl	opt	cmd

German QWERTZ keyboard.

²	&	É	“	’	(-	È	_	Ç	À)	=	delete
tab	A	Z	E	R	T	Y	U	I	O	P	^	\$	enter
caps	Q	S	D	F	G	H	J	K	L	M	Ù	*	
shift	\	W	X	C	V	B	N	,	;	:	!	shift	
cntrl	opt	cmd	space								cntrl	opt	cmd

French AZERTY keyboard.

How Is Type Input?

Dvorak Simplified

In 1936, Dr. August Dvorak and his brother-in-law, Dr. William Dealey, patented a new visual layout for keyboards. Dvorak and his team as well as the American National Standards Institute (ANSI) continued to develop variations on the original design. Known today as the Dvorak layout, it is the only visual layout registered with ANSI other than QWERTY.

Proponents claim the Dvorak layout requires less finger motion, reduces errors, and increases typing speed in comparison to QWERTY. More recently, Dvorak layouts are purported to reduce repetitive motion injuries such as carpal tunnel syndrome. The Dvorak layout supposedly accomplishes all this because most commonly used letters are placed in the home row.

Dvorak adoption has been slow in spite of shipping as an option with all major operating systems. There are several contributing factors: studies on the benefits of Dvorak have been inconclusive; the cost of replacing QWERTY layout keyboards is too high; and re-learning how to type is just too hard.

Key stroke frequency distribution (English)

Row	QWERTY	Dvorak
Top	52%	22%
Home	32%	70%
Bottom	16%	8%

`	1	2	3	4	5	6	7	8	9	0	-	=	delete
tab	Q	W	E	R	T	Y	U	I	O	P	[]	\
caps	A	S	D	F	G	H	J	K	L	;	'		enter
shift		Z	X	C	V	B	N	M	,	.	/		shift
cntrl	opt	cmd	space							cntrl	opt	cmd	

United States QWERTY keyboard.

`	1	2	3	4	5	6	7	8	9	0	[]	delete
tab	'	,	.	P	Y	F	G	C	R	L	/	=	\
caps	A	O	E	U	I	D	H	T	N	S	-		enter
shift		;	Q	J	K	X	B	M	W	V	Z		shift
cntrl	opt	cmd	space							cntrl	opt	cmd	

American Dvorak Simplified keyboard layout.

How Is Type Input?

Key Strokes to Words

The connection between keyboard and displayed text seems pretty direct – you press a key and a letter appears.

Interpreting a mechanical key press to mean that the user wants to input a specific character or mark is actually a multi-step process. Each mechanical key has an associated raw keycode. It is the keycode, not the letter, that’s passed to the device OS when a key is pressed. The keyboard’s functional layout converts the keycode to a character-encoding hex code (typically Unicode). Each hex code has a unique character association. That character is then passed to the current application which then renders the associated letterform glyph.

Keyboard
The keycodes for each mechanical key are shown in magenta

50	18	19	20	21	23	22	26	28	25	29	27	24	51	
`	1	2	3	4	5	6	7	8	9	0	-	=	delete	
48		12	13	14	15	17	16	32	34	31	35	33	30	42
tab		Q	W	E	R	T	Y	U	I	O	P	[]	\
57		0	1	2	3	5	4	38	40	37	41	39	36	
caps		A	S	D	F	G	H	J	K	L	;	'	return	
56		6	7	8	9	11	45	46	43	47	44	56		
shift		Z	X	C	V	B	N	M	,	.	/	shift		
59	58	55	space								55	58	59	
cntrl	opt	cmd									cntrl	opt	cmd	

Mechanical Keys

shift							shift						shift
H	E	L	L	O	,	space	W	O	R	L	D	1	

Keycodes

56-4	14	37	37	31	43	49	56-13	31	15	37	2	56-18	
------	----	----	----	----	----	----	-------	----	----	----	---	-------	--

Unicode Values

This is how text information is stored in computer files

0048	0065	006C	006C	006F	002C	00A0	0057	006F	0072	006C	0064	0021	
------	------	------	------	------	------	------	------	------	------	------	------	------	--

Characters

H	e	l	l	o	,		W	o	r	l	d	!	
---	---	---	---	---	---	--	---	---	---	---	---	---	--

Displayed Words

Hello, World!

Mechanical keys pressed and raw keycodes sent to keyboard driver.

Keycodes translated into Unicode values by the functional layout.

Unicode values used to generate character string. (See page 48.)

Character string output on display (specific formatting choices such as font and type size are applied before final onscreen display).

How Is Type Input?

Language Support

The user controls the language of the operating system and keyboard functional layout. The author of a document chooses what language to write in. Any given operating system must be able to display and allow the editing of any language in any editable document.

Choosing an operating system language should change the text on all user interface elements (menus, buttons, window titles, etc.), the keyboard functional layout, and display defaults for date, time formats, monetary units, and measurements such as temperature, distance, weight, and volume. The user can override language defaults and choose keyboard functional layouts and data display options for other languages.

Text in documents is encoded as a series of numeric values with each number uniquely assigned to a letter, figure, or symbol from virtually any writing system. (See page 18 and the next section.) Metadata in the document tells the application layout manager and OS screen renderer what font to use to display the encoded letters, figures, or symbols.

No font contains glyphs for all possible letters or symbols. The screen renderer displays a placeholder character (typically a rectangle) from the font in lieu of any missing glyphs.



Language selection in the Mac OS

Typeface: Monaco

I am testing how to write in multiple languages.

Ш фь еуыештп рщц ещ цкшеу шт ьгдешзду дфтпгфпуую

□ □□ □□ □□ □ □□□ □□ □□□ □□□□.

← A placeholder character is substituted for missing glyphs.

Typeface: Arial Unicode

I am testing how to write in multiple languages.

Ш фь еуыештп рщц ещ цкшеу шт ьгдешзду дфтпгфпуую

이 ㅣㅁ 텃팅 호ㅡ 토 ㅡ리터 ㅣㄴ 물뿔러 란구ㅣ것.

← The same text displays properly when set in a font with appropriate glyphs.

How Is Type Input?

Input Method Editors

Mechanical keyboards are efficient text input devices for most script systems. Primary characters are each assigned a key and alternate versions (e.g. capitals), special characters, and diacritical marks are accessed by using modifier keys (e.g. Shift, Alt, and Command) in combination with the primary keys. This effectively quadruples the number of characters that can be input with a mechanical keyboard. Script systems such as Japanese, Chinese, and Korean use far more characters than could be accessed directly with a mechanical keyboard. An additional step, known as an input method editor (IME) is required.

For example, Japanese text is typically entered in one of two ways: through “romaji” (transliterated Japanese syllables in a romanized form) that is then converted by the computer into kana (Japanese syllabary), or typed directly in kana. Once kana are input, they are either left as is or converted to kanji (Chinese characters). The Japanese language has many homophones, and conversion of a kana spelling (representing the pronunciation) into a kanji (representing the standard written form of the word) is often a one-to-many process. A kana to kanji converter offers a list of candidate kanji characters for the input kana, and the user may use the space bar or arrow keys to scroll through the list of candidates until he or she reaches the correct one. Sophisticated input method editors attempt to guess for the user based on context.

Chinese text also can be entered in multiple ways. The first is nearly identical to the Japanese romaji method and is called “pinyin”. The other way to enter Chinese text is shape-based and comes in many forms (Cangjie and Wubi are the most popular). The system uses 26 “radicals” that are the basic shapes of the Chinese script. The method requires the typist be familiar with several decomposition rules that define how to analyze a character to arrive at its key sequence. These methods are considered much faster for experienced typists since each character is visually unique and therefore will have a unique key sequence (thus preventing the need to scroll through a list of possible characters) but also more difficult to learn for the same reason.

Korean is typically entered in a manner very similar to Chinese shape methods.

Romanized Input: Japanese Romaji Method

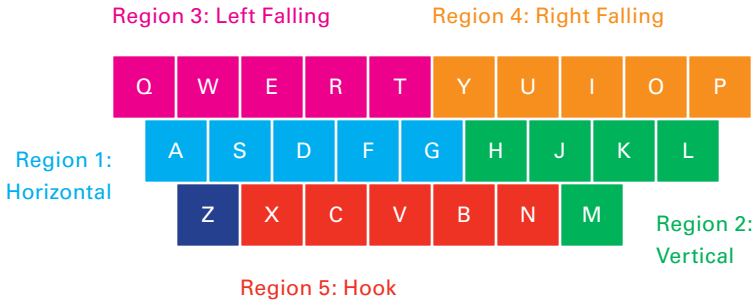
Typing in Japanese on a western ANSI keyboard requires the use of an input editor to select the proper kana and kanji. In the example below, the words *gokurousama deshita* (thank you for your hard work) are entered on an iPhone.



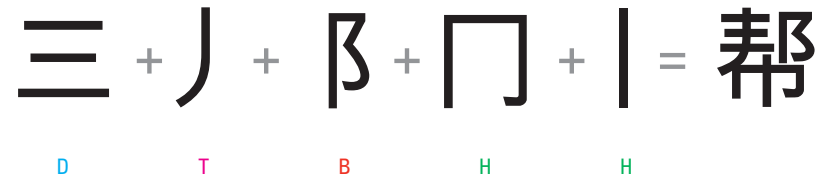
While typing, the editor presents the user with a short-list of options for what the kanji conversion could be. Once the word is typed in, the user can click forward to get a full palette of options to select the correct word.

Shape-based Input: Chinese Wubi Method

Every Chinese character can be broken down into its root characters. Conversely, characters can be built by combining these roots. The Wubi keyboard functional layer has five regions, organized by the direction of the root’s first stroke. Each region contains five keys, and each key can access multiple roots in combination with the Shift, Alt, and Command keys. The diagram below shows the regions of the Wubi keyboard mapped against the ANSI keyboard layout.



In the example below, the word 帮 is built from components. The matching ANSI keys are shown below each Chinese root character. One interesting aspect of the Wubi method is that it is not always necessary to type all of the roots of a character. In the example below, the typist would not press DTBHH, but instead only DTBH because there are no characters in Chinese that are formed by pressing DTBH with any other final root. There are many other examples of this, including quite a few characters with four- or five-root elements that can be typed with only two keystrokes.



How Is Type Encoded?

Character encoding systems pair each supported character with a unique ID number or code point. These ID numbers tell the current application what glyph to display on screen. The number of characters that could be included in any encoding system is represented in terms of “bits”. A bit is the smallest unit of digital information and can have a value of “1” or “0”. Though the term “bit” wasn’t used in the pre-digital era it is still an effective means of expressing the capabilities of early encoding systems. In the case of the older technology a “bit” measures states such as on/off (e.g. digits), up/down (e.g. key positions) or long/short (e.g. sounds). The character count for any encoding system is determined by raising the number 2 to the number of bits. A 5-bit system, therefore, could encode 32 letters ($2 \times 2 \times 2 \times 2 \times 2 = 32$).

Over the past 30 years, multiple encoding systems have been defined and subsequently superseded as the need for more unique character IDs has grown in response to increased international document exchange.

The ASCII encoding system handled only 128 characters (7-bit). Today, the contemporary standard, Unicode, can handle up to 1,114,112 possible characters (though no one has come close to needing that large a set, yet).

Year	Standard	Size	Code points
1836	Morse Code	–	36+
1874	Baudot Code	5-bit	32
1901	Murray Code	5-bit (×2)	64
1956	Fieldata	6-bit	64
1963	ASCII	7-bit (½ 8-bit)	128
1984	Mac OS Roman	8-bit	256
1986	ISO/IEC 8859	8-bit (×16 pages)	4,096
1987	Unicode 1	16-bit (double byte)	65,563
1996	Unicode 2	21-bit	1,114,112

How Is Type Encoded?

Counting for Computers

Most people count using the decimal, or base-10, number system.

In the decimal number system there are ten possible values for any digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Counting in decimal begins with 0 and proceeds through 9 before moving on the next column. When the values for the first column are exhausted, the next-higher column (to the left) is incremented, and counting starts over at 0.

Computers count in the binary, or base-2, number system. In the binary number system there are only two possible values for any digit: 0 or 1. Counting in binary is similar to counting in decimal. Beginning with a single digit, counting proceeds through each value, in increasing order. When the values for the first column are exhausted, the next-higher column (to the left) is incremented, and counting starts over at 0.

Hexadecimal is a base-16 number system. In the hexadecimal number system, there are 16 values for any digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Computers encode data into units of information called bytes. Most bytes consist of eight bits, which is a convenient power of two permitting values from 0 to 255 for one byte.

Most computers manipulate binary data; however it is difficult for humans to work with the large number of digits for even relatively small binary numbers. Although most humans are familiar with decimal numbering, it is much easier to map binary to hexadecimal than to decimal, because each hexadecimal digit maps evenly to one 8-bit byte.

Decimal (base-10)	Binary (base-2)	Hexadecimal (base-16)
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

The second column indicates the number of tens for base-10, the number of twos for base-2, and the number of sixteens for base-16. Subsequent columns are powers of the base, e.g. the third column of base-2 is two to the second power or the fours column.

8-bit Byte to Hexadecimal

1 byte has 8 bits*
has values 0 – 255 → in binary: 0000 0000 – 1111 1111

can be divided into 4 bits + 4 bits

4 bits has values 0 – 15

equals 1 hexadecimal digit

1 byte can be expressed with 2 hexadecimal digits
0000 0000 – 1111 1111 → 00 – FF

(This system proves quite convenient in specifying characters in font tables and colors in color spaces, e.g. white = FFFFFFFF.)

*
A byte does not have to be 8 bits, it can be any number of bits. However, 8 bits is the *de facto* standard.

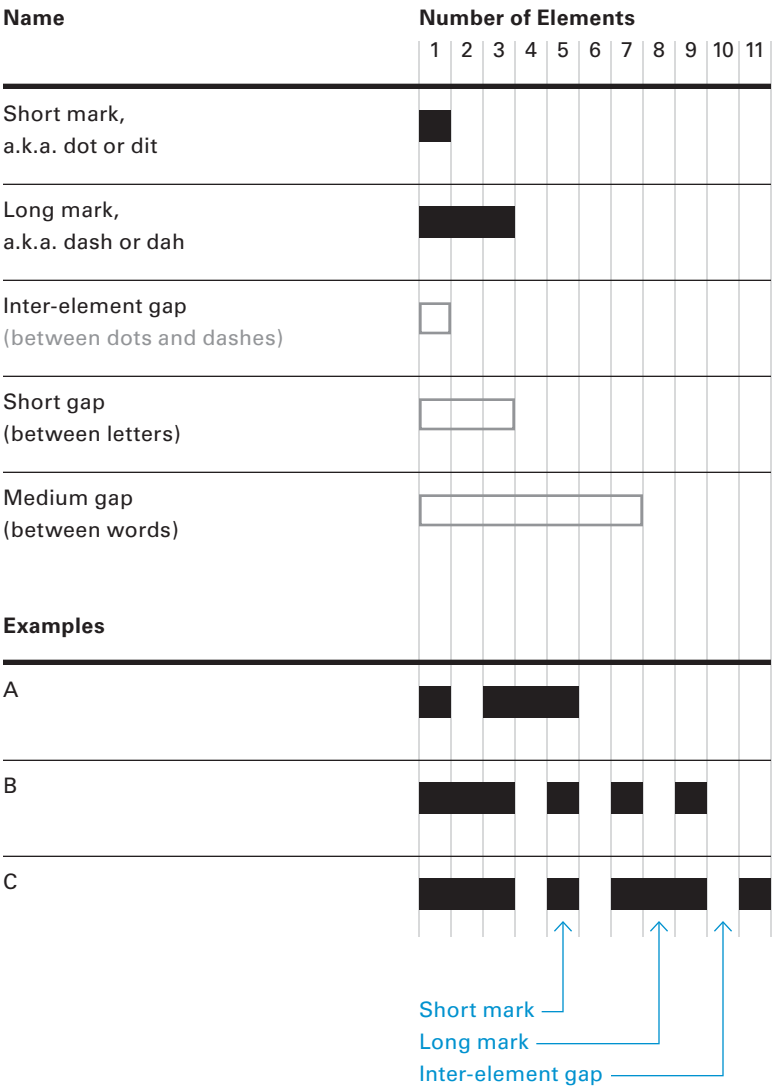
← Binary and hexadecimal map neatly to each other: four binary digits (4 bits) has the same value as one hexadecimal digit.

How Is Type Encoded?

Morse Code

In 1836, Samuel B. Morse, Joseph Henry, and Alfred Vail began to develop a system for sending text via telegraph that eventually would become known as Morse Code. The system **transmits text as a series of on-off tones, lights, or clicks** that can be directly understood by a skilled listener or observer without special equipment. The International Morse Code encodes the Roman alphabet, the Arabic numerals and a small set of punctuation and procedural signals as standardized sequences of short and long “dots” and “dashes”, or “dits” and “dahs”. Extensions to the Morse alphabet exist for non-English script systems that require more than the basic set of 26 letters.

Morse code speed is specified in words per minute (WPM) and associated with an “element time” equal to 1.2 seconds divided by the speed in WPM. A dot consists of an “on” element followed by an “off” element, and a dash is three “on” elements and one “off” element. **Each character is a sequence of dots and dashes, with the shorter sequences assigned to the more frequently used letters in English – the letter “E” is represented by a single dot, and the letter “T” by a single dash.** A speed of 12 WPM is therefore associated with an element time of 100 milliseconds, so each dot is 100 ms long and each dash is 300 ms long, each followed by 100 ms of silence.



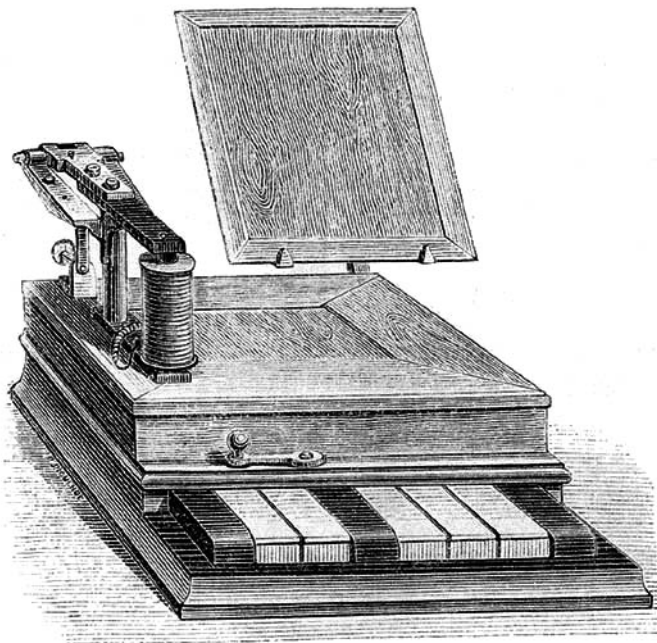
Code	Character
. —	A
— ...	B
— . — .	C
— ..	D
.	E
. — .	F
— — .	G
....	H
..	I
. — — —	J
— . —	K
. — . .	L
— —	M
— .	N
— — —	O
. — — .	P
— — — —	Q
. — .	R
...	S
—	T
. — —	U
... —	V
. — — —	W
— .. —	X
— . — —	Y
— — . .	Z

Code	Character
— — — — —	0
. — — — —	1
.. — — —	2
... — —	3
.... —	4
.....	5
—	6
— — ...	7
— — — .	8
— — — — .	9
. — . — .	.
— . — . — —	,
.. — — .	?
. — — — — .	'
. — . — — —	!
— . — . — —	/
— . — — .	(
— . — — — —)
. — ...	& (also: Wait)
— — — . .	:
— . — . — .	;
— . . . —	=
. — . — .	+
—	-
.. — — — —	_
. — . — .	"
... — — — —	\$
. — . — . .	@

How Is Type Encoded?

Baudot Code

Émile Baudot invented his eponymous 5-bit coding system in 1874. Later known as the International Telegraph Alphabet No.1 (ITA1), the code was entered on a five key, piano-like keyboard. The left hand controlled two keys, the right the other three. Once the keys had been pressed they were locked down until mechanical contacts in a distributor unit passed over the sector connected to that particular keyboard, when the keyboard was unlocked ready for the next character to be entered, with an audible click (known as the “cadence signal”) to warn the operator. Operators had to maintain a steady rhythm, and the usual speed of operation was 30 words per minute. The operator had to memorize the five-unit codes. Received messages were printed or punched onto paper tape.



A Baudot keyboard.

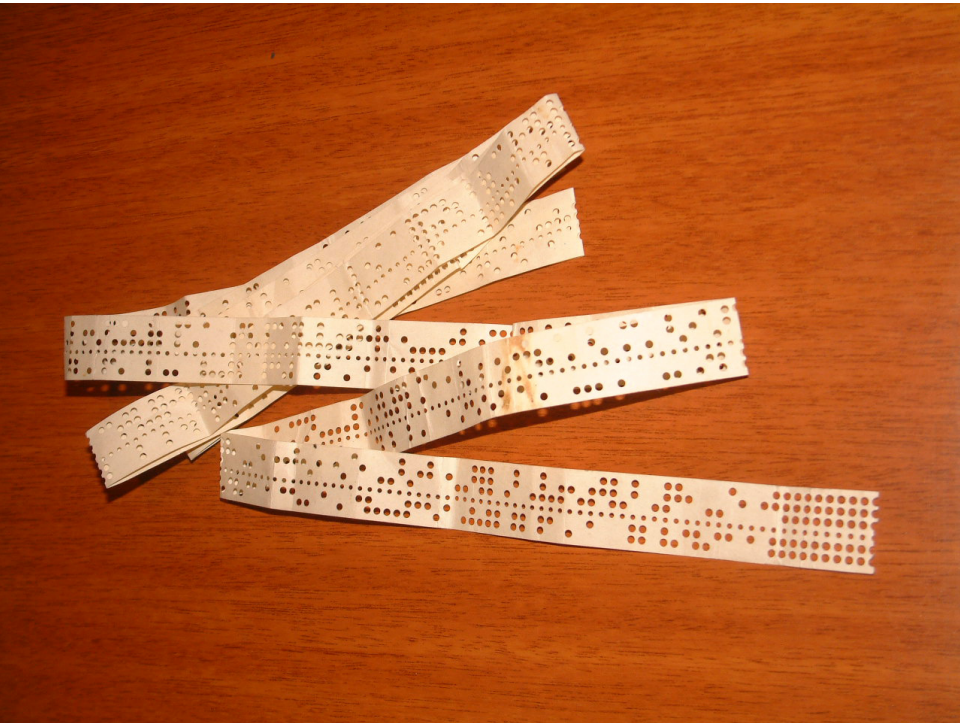
Pattern of Impulses		Letters	Figures
1 = key press 0 = no key press Left Right			
00	100	A	1
01	001	B	8
01	101	C	9
01	111	D	0
00	010	E	2
00	110	E'	&
01	011	F	f
01	010	G	7
01	110	H	h
00	011	I	o
01	100	J	6
11	100	K	(
11	110	L	=
11	010	M)
11	011	N	N°
00	111	O	5
11	111	P	%
11	101	Q	/
11	001	R	-
10	001	S	;
10	101	T	!
00	101	U	4
10	111	V	'
10	011	W	?
10	010	X	,
00	001	Y	3
10	110	Z	:
10	100	t	.
11	000	Erasure	Erasure
01	000	Shift to figures	
10	000	Shift to letters	

How Is Type Encoded?

Murray Code

In 1901 the Baudot code was modified by Donald Murray, prompted by the development of a typewriter-like keyboard. This revised code is sometimes known as the Baudot-Murray code, and later (1930) would be revised slightly to become the International Telegraphy Alphabet 2 (ITA2). It employed a “keyboard perforator” that punched a paper tape when the keys were pressed. Because there was no longer a direct correlation between the operator’s hand movements and the bits transmitted, there was no concern about arranging the keys to minimize hand fatigue, instead Murray designed the keyboard layout to minimize wear on the machinery, assigning the code combinations with the fewest punched holes to the most frequently used characters. Received messages were printed or punched onto paper tape.

Like the Baudot code, the Murray code is a 5-bit encoding system (32 characters), however the Murray code makes use of a shift key/code that allows the user to switch to a second set of 32 characters (letters in one set, numbers and punctuation in the other). The Murray code also introduced control characters for the first time, such as CR (carriage return) and LF (line feed).



A Murray tape.

Pattern of Impulses 1 = mark 0 = space	Letters	Figures
00000	null	null
00100	space	space
11101	Q	1
11001	W	2
10000	E	3
01010	R	4
00001	T	5
10101	Y	6
11100	U	7
01100	I	8
00011	O	9
01101	P	0
11000	A	-
10100	S	'
10010	D	\$
10110	F	!
01011	G	&
00101	H	#
11010	J	'
11110	K	(
01001	L)
10001	Z	"
10111	X	/
01110	C	:
01111	V	;
10011	B	?
00110	N	,
00111	M	.
00010	Carriage return	Carriage return
01000	Line feed	Line feed
11011	Shift to figures	
11111		Shift to letters

How Is Type Encoded?

ASCII

In 1963, the X3 committee of the American Standards Association introduced the American Standard Code for Information, better known as ASCII. A 7-bit character encoding system, ASCII is based on the order of the English alphabet. Most modern character-encoding systems are based on ASCII though they support far more characters. ASCII includes definitions for 128 characters: 33 are non-printing control characters (now mostly obsolete) that affect how text and space is processed; 94 are printable characters, and the space is considered an invisible graphic.

Unicode values for each character are listed in gray below the character.

The ASCII character map is arranged so that the column and row together form the hexadecimal number of each character.

For instance: A = 4 1



Almost all of the characters in the ASCII table are mapped to a key, or combination of keys, on a keyboard via the functional layout. (See page 14.) Control characters, indicated in blue, do not map to any keys. When a key is pressed, what is actually transmitted to the operating system is a binary value.

	0	1	2	3	4	5	6	7
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076
7	BEL 0007	ETB 0017	' 0027	7 0037	G 0047	W 0057	g 0067	w 0077
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F

How Is Type Encoded?

Codepages

The evolution of computers from, broadly speaking, number-crunchers to information processors was hampered by the limited ASCII characters set. Only 94 of 128 characters were printable characters, and these were taken exclusively from the Latin script. This severely limited the languages that could be supported with ASCII because most languages that use the Latin alphabet require additional characters not used in English such as ß (German), ñ (Spanish), and å (Swedish and other Nordic languages).

In order to add characters needed for other languages, a variety of 8-bit standards were developed. The 256-character sets are typically represented as 16 x 16 tables, taking advantage of hex code. The tables are called codepages.

The first 128 code points reproduce the original 7-bit ASCII system. (The eighth bit of a byte had previously be used for data transmission protocol information or was left unused to save space.)

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

The Mac OS Roman codepage, one of the first 8-bit extensions to ASCII, developed in 1984.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	Ä 00C4	è 00EA	† 2020	∞ 221E	¿ 00BF	— 2013	‡ 2021	🍏 F8FF
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	Å 00C5	ë 00EB	° 00B0	± 00B1	ì 00A1	— 2014	· 00B7	Ò 00D2
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	Ç 00C7	í 00ED	ø 00A2	≤ 2264	¬ 00AC	“ 201C	, 201A	Ú 00DA
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	É 00C9	ì 00EC	£ 00A3	≥ 2265	√ 221A	” 201D	„ 201E	Û 00DB
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	Ñ 00D1	î 00EE	§ 00A7	¥ 00A5	f 0192	‘ 2018	‰ 2030	Ù 00D9
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	Ö 00D6	ï 00EF	• 2022	μ 00B5	≈ 2248	’ 2019	Â 00C2	ı 0131
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	Ü 00DC	ñ 00F1	¶ 00B6	∂ 2202	Δ 2206	÷ 00F7	Ê 00CA	ˆ 02C6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	á 00E1	ó 00F3	ß 00DF	Σ 2211	« 00AB	◊ 25CA	Á 00C1	˜ 02DC
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	à 00E0	ò 00F2	® 00AE	Π 220F	» 00BB	ÿ 00FF	Ë 00CB	— 00AF
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	â 00E2	ô 00F4	© 00A9	π 03C0	… 2026	Ÿ 0178	È 00C8	˘ 02D8
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	ä 00E4	ö 00F6	™ 2122	∫ 222B	NBSP 00A0	/ 2044	Í 00CD	· 02D9
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	ä 00E3	õ 00F5	’ 00B4	ª 00AA	À 00C0	€ 20AC	Î 00CE	° 02DA
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	å 00E5	ú 00FA	“ 00A8	º 00BA	Ã 00C3	‹ 2039	Ï 00CF	, 00B8
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	ç 00E7	ù 00F9	≠ 2260	Ω 03A9	Ö 00D5	› 203A	ì 00CC	“ 02DD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	é 00E9	û 00FB	Æ 00C6	æ 00E6	Œ 0152	fi FB01	Ó 00D3	˚ 02DB
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	è 00E8	ü 00FC	Ø 00D8	ø 00F8	œ 0153	fl FB02	Ô 00D4	˘ 02C7

Identical to ASCII

Extension for non-English languages

How Is Type Encoded?

Windows 1252

Windows-1252 or CP-1252 is a Latin alphabet character encoding system, used by default in legacy components of Microsoft Windows in English and some other Western languages. It is one of many Windows codepages and includes almost all of the same characters as Mac OS Roman codepage but assigns them to different positions.

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	€ 20AC		NBSP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	đ 00F0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071		‘ 2018	¡ 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	, 201A	’ 2019	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	f 0192	“ 201C	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	„ 201E	” 201D	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	… 2026	• 2022	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	ö 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	† 2020	– 2013	¡ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	‡ 2021	— 2014	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	^ 02C6	˘ 02DC	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	% 2030	™ 2122	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	Š 0160	š 0161	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	‹ 2039	› 203A	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	Œ 0152	œ 0153	¬ 00AC	¼ 00BC	ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D			SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	Ž 017D	ž 017E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F		Ÿ 0178	¯ 00AF	¿ 00BF	İ 00CF	ß 00DF	ï 00EF	ÿ 00FF

Identical to ASCII

Positions 81, 8D, 8F, 90, and 9D are unused.

How Is Type Encoded?

ISO/IEC 8859

ISO/IEC 8859 was designed in the mid-1980s by the European Computer Manufacturer’s Association (ECMA). Later, development of the codepage layouts was taken over by the International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). The project was an attempt to create a universal standard for code pages. Over time it grew to include 16 codepages, enough to cover all Latin-based scripts and then some. However it never addressed Chinese, Japanese, or Korean scripts (CJK). It was a precursor to the Unicode standard.

Between all 16 codepages, ISO/IEC 8859 covered 2,176 possible code points – the first 128 code points of each codepage are identical to ASCII while the latter 128 are at least partially unique in each codepage. There are many shared characters between the codepages of ISO 8859. Characters identical to ISO 8859-1 are marked in all subsequent codepages.

Codepage 1	ISO/IEC 8859-1	Latin-1, Western European	Perhaps the most widely used part of ISO/IEC 8859, covering most Western European languages: Danish (partial), Dutch (partial), English, Faeroese, Finnish (partial), French (partial), German, Icelandic, Irish, Italian, Norwegian, Portuguese, Rhaeto-Romanic, Scottish Gaelic, Spanish, and Swedish. Languages from other parts of the world are also covered, including Eastern European Albanian, Southeast Asian Indonesian, as well as the African languages Afrikaans and Swahili.
Codepage 2	ISO/IEC 8859-2	Latin-2, Central European	Supports those Central and Eastern European languages that use the Latin alphabet but need different diacritics compared to the Latin-1 set, including Bosnian, Polish, Croatian, Czech, Slovak, Slovene, Serbian, and Hungarian.
Codepage 3	ISO/IEC 8859-3	Latin-3, Southern European	Turkish, Maltese, and Esperanto.
Codepage 4	ISO/IEC 8859-4	Latin-4, Northern European	Estonian, Latvian, Lithuanian, Greenlandic, and Sami.
Codepage 5	ISO/IEC 8859-5	Latin / Cyrillic	Covers mostly Slavic languages that use a Cyrillic alphabet including Belarusian, Bulgarian, Macedonian, Russian, Serbian, and Ukrainian (partial).
Codepage 6	ISO/IEC 8859-6	Latin / Arabic	Covers the most common Arabic language characters. Doesn’t support other languages using the Arabic script.
Codepage 7	ISO/IEC 8859-7	Latin / Greek	Covers the modern Greek language (monotonic orthography). Can also be used for Ancient Greek written without accents or in monotonic orthography, but lacks the diacritics for polytonic orthography.
Codepage 8	ISO/IEC 8859-8	Latin / Hebrew	Covers the modern Hebrew alphabet as used in Israel.
Codepage 9	ISO/IEC 8859-9	Latin-5, Turkish	Largely the same as ISO/IEC 8859-1, replacing the rarely used Icelandic letters with Turkish ones. It is also used for Kurdish.
Codepage 10	ISO/IEC 8859-10	Latin-6, Nordic	A rearrangement of Latin-4. Considered more useful for Nordic languages. Baltic languages use Latin-4 more.
Codepage 11	ISO/IEC 8859-11	Latin / Thai	Contains characters needed for the Thai language. Virtually identical to TIS 620.
Codepage 12	ISO/IEC 8859-12	Latin / Devanagari	Officially abandoned in 1997.
Codepage 13	ISO/IEC 8859-13	Latin-7, Baltic Rim	Added some characters for Baltic languages which were missing from Latin-4 and -6.
Codepage 14	ISO/IEC 8859-14	Latin-8, Celtic	Covers Celtic languages such as Gaelic and the Breton language.
Codepage 15	ISO/IEC 8859-15	Latin-9	A revision of 8859-1 that completes the coverage of French, Finnish, and Estonian.
Codepage 16	ISO/IEC 8859-16	Latin-10, South Eastern European	Intended for Albanian, Croatian, Hungarian, Italian, Polish, Romanian, and Slovene, but also covers Finnish, French, German, and Irish Gaelic (new orthography).

Languages listed as “partial” are not fully expressible with that particular codepage.

How Is Type Encoded?

ISO/IEC 8859-1

Latin-1 / Western European

Note that in all of the ISO/IEC 8859 codepages, positions 0080 – 009F are used for control characters that were not present in Mac OS Roman or the Windows codepages. Often, these code points will be shown as blank or empty in representations of the 8859 pages, which is confusing and potentially misleading since they are not actually empty, rather they are used for non-printing characters.

This portion of the ISO/IEC 8859 encoding system has been carried forward as the first 256 code points of the Unicode standard – the current standard for font encoding.

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	đ 00F0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	ı 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Þ 00DE	î 00EE	þ 00FE
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	— 00AF	¿ 00BF	Ĭ 00CF	Ɓ 00DF	ĩ 00EF	ÿ 00FF

Identical to ASCII

How Is Type Encoded?

ISO/IEC 8859-2

Latin-2 / Central European

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	Ř 0154	Đ 0110	í 0155	ď 0111
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ą 0104	ą 0105	Á 00C1	Ń 0143	á 00E1	ń 0144
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	ˆ 02D8	˘ 02DB	Â 00C2	Ň 0147	â 00E2	ň 0148
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	Ł 0141	ł 0142	Ă 0102	Ó 00D3	ă 0103	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	Ĺ 013D	ĺ 013E	Ľ 0139	Õ 0150	í 013A	ċ 0151
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Š 015A	š 015B	Ć 0106	Ö 00D6	ć 0107	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	Ş 00A7	˘ 02C7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	‚ 00B8	Č 010C	Ř 0158	č 010D	ř 0159
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	Š 0160	š 0161	É 00C9	Û 016E	é 00E9	û 016F
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Ş 015E	ş 015F	Ę 0118	Ú 00DA	ę 0119	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	ť 0164	ť 0165	Ě 00CB	Ů 0170	ě 00EB	ů 0171
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	Ž 0179	ž 017A	Ě 011A	Ü 00DC	ě 011B	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	˘ 02DD	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	Ž 017D	ž 017E	Î 00CE	Ț 0162	î 00EE	ț 0163
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Ž 017B	ž 017C	Ď 010E	ß 00DF	ď 010F	· 02D9

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-3

Latin-3 / South European

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 00C0		à 00E0	
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ĥ 0126	ħ 0127	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	˘ 02D8	² 00B2	Â 00C2	Ô 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3		Ó 00D3		ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ö 00D4	ä 00E4	ö 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095		µ 00B5	Ć 010A	Ġ 0120	ć 010B	ġ 0121
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Ħ 0124	ħ 0125	Č 0108	Ö 00D6	č 0109	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ê 011C	è 00E8	ê 011D
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	İ 0130	ı 0131	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Ş 015E	ş 015F	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	Ğ 011E	ğ 011F	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	Ĵ 0134	ĵ 0135	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	Ŭ 016C	í 00ED	ŭ 016D
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E			Î 00CE	Ŝ 015C	î 00EE	ŝ 015D
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Ž 017B	ž 017C	Ĭ 00CF	Ɓ 00DF	ĩ 00EF	· 02D9

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-4

Latin-4 / Northern European

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	Ā 0100	Đ 0110	ā 0101	đ 0111
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ą 0104	ą 0105	Á 00C1	Ŋ 0145	á 00E1	ŋ 0146
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	κ 0138	ς 02DB	Â 00C2	Ō 014C	â 00E2	ō 014D
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	Ŕ 0156	ŕ 0157	Ã 00C3	Ɔ 0136	ã 00E3	ķ 0137
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	▯ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	Ĭ 0128	ĩ 0129	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Ł 013B	ł 013C	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	Š 00A7	ˇ 02C7	Ĺ 012E	× 00D7	š 012F	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	‚ 00B8	Č 010C	Ø 00D8	č 010D	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	Š 0160	š 0161	É 00C9	Ȳ 0172	é 00E9 233	ȳ 0173
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Ě 0112	ě 0113	Ė 0118	Ú 00DA	ę 0119	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	Ģ 0122	ģ 0123	Ď 00CB	Ů 00DB	ď 00EB	ů 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	ƒ 0166	ƒ 0167	Ê 0116	Ü 00DC	ê 0117	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	Ŋ 014A	Í 00CD	Ŭ 0168	í 00ED	ŭ 0169
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	Ž 017D	ž 017E	Î 00CE	Ű 016A	î 00EE	ű 016B
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	– 00AF	ŋ 014B	Ī 012A	ß 00DF	ī 012B	· 02D9

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-5

Latin / Cyrillic

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	А 0410	Р 0420	а 0430	р 0440	№ 2116
1	SOH 0001	DC1 0011	! 0021	1 0031	А 0041	Q 0051	а 0061	q 0071	HOP 0081	PU1 0091	Ё 0401	Б 0411	С 0421	б 0431	с 0441	ё 0451
2	STX 0002	DC2 0012	“ 0022	2 0032	В 0042	Р 0052	в 0062	р 0072	BPH 0082	PU2 0092	Ђ 0402	В 0412	Т 0422	в 0432	т 0442	ђ 0452
3	ETX 0003	DC3 0013	# 0023	3 0033	С 0043	S 0053	с 0063	s 0073	NBH 0083	STS 0093	Ѓ 0403	Г 0413	У 0423	г 0433	у 0443	ѓ 0453
4	EOT 0004	DC4 0014	\$ 0024	4 0034	Д 0044	Т 0054	д 0064	т 0074	IND 0084	CCH 0094	Є 0404	Д 0414	Ф 0424	д 0434	ф 0444	є 0454
5	ENQ 0005	NAK 0015	% 0025	5 0035	Е 0045	U 0055	е 0065	u 0075	NEL 0085	MW 0095	Ѕ 0405	Е 0415	Х 0425	е 0435	х 0445	ѕ 0455
6	ACK 0006	SYN 0016	& 0026	6 0036	Ѓ 0046	Ѕ 0056	ѓ 0066	џ 0076	SSA 0086	SPA 0096	Ї 0406	Ж 0416	Ц 0426	ж 0436	ц 0446	ї 0456
7	BEL 0007	ETB 0017	‘ 0027	7 0037	Г 0047	W 0057	г 0067	w 0077	ESA 0087	EPA 0097	Ї 0407	З 0417	Ч 0427	з 0437	ч 0447	ї 0457
8	BS 0008	CAN 0018	(0028	8 0038	Н 0048	Х 0058	н 0068	х 0078	HTS 0088	SOS 0098	Ј 0408	И 0418	Ш 0428	и 0438	ш 0448	ј 0458
9	HT 0009	EM 0019) 0029	9 0039	Ї 0049	Y 0059	і 0069	y 0079	HTJ 0089	SGCI 0099	Љ 0409	Й 0419	Щ 0429	й 0439	щ 0449	љ 0459
A	LF 000A	SUB 001A	* 002A	: 003A	Ј 004A	Z 005A	ј 006A	z 007A	VTs 008A	SCI 009A	Њ 040A	К 041A	Ђ 042A	к 043A	ђ 044A	њ 045A
B	VT 000B	ESC 001B	+ 002B	; 003B	К 004B	[005B	к 006B	{ 007B	PLD 008B	CSI 009B	Ћ 040B	Л 041B	Ы 042B	л 043B	ы 044B	ћ 045B
C	FF 000C	FS 001C	, 002C	< 003C	Л 004C	\ 005C	л 006C	 007C	PLU 008C	ST 009C	Ќ 040C	М 041C	Ь 042C	м 043C	ь 044C	ќ 045C
D	CR 000D	GS 001D	- 002D	= 003D	М 004D	Ј 005D	м 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	Н 041D	Э 042D	н 043D	э 044D	§ 00A7
E	SO 000E	RS 001E	. 002E	> 003E	Н 004E	^ 005E	н 006E	~ 007E	SS2 008E	PM 009E	Ў 040E	О 041E	Ю 042E	о 043E	ю 044E	ў 045E
F	SI 000F	US 001F	/ 002F	? 003F	О 004F	_ 005F	о 006F	DEL 007F	SS3 008F	APC 009F	Ў 040F	П 041F	Я 042F	п 043F	я 044F	ѵ 045F

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-6

Latin / Arabic

Designed to cover languages using the Arabic alphabet. Only nominal letters are encoded, no preshaped forms of the letters, so shaping processing is required for display. It was never very popular.

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0			ذ 0630	- 0640	ﻻ 0650
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091			ء 0621	ر 0631	ف 0641	ﻻ 0651
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092			آ 0622	ز 0632	ق 0642	ﻻ 0652
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093			إ 0623	س 0633	ك 0643	
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	؀ 00A4		ؤ 0624	ش 0634	ل 0644	
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095			إ 0625	ص 0635	م 0645	
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096			ئ 0626	ض 0636	ن 0646	
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097			ا 0627	ط 0637	ه 0647	
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098			ب 0628	ظ 0638	و 0648	
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099			ة 0629	ع 0639	ى 0649	
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A			ت 062A	غ 063A	ي 064A	
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B		؛ 061B	ث 062B		ﻻ 064B	
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	، 060C		ج 062C		ﻻ 064C	
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD		ح 062D		ﻻ 064D	
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E			خ 062E		ﻻ 064E	
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F		؟ 061F	د 062F		ﻻ 064F	

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-7

Latin / Greek

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	ı̇ 0390	Π 03A0	ŭ 03B0	π 03C0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	‘ 2018	± 00B1	À 0391	Ʋ 03A1	α 03B1	ρ 03C1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	’ 2019	² 00B2	Β 0392		β 03B2	ς 03C2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3	Γ 0393	Σ 03A3	γ 03B3	σ 03C3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	€ 20AC	´ 0384	Δ 0394	Ƣ 03A4	δ 03B4	τ 03C4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	□ 20AF	¨ 0385	Ε 0395	Υ 03A5	ε 03B5	υ 03C5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	ı̇ 00A6	À 0386	Z 0396	Φ 03A6	ζ 03B6	φ 03C6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	H 0397	X 03A7	η 03B7	χ 03C7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	Ǝ 0388	Θ 0398	Ψ 03A8	θ 03B8	ψ 03C8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	Ɔ 0389	Ι 0399	Ω 03A9	ι 03B9	ω 03C9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ˆ 037A	ı̇ 038A	K 039A	İ 03AA	κ 03BA	ı̇ 03CA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Λ 039B	Ÿ 03AB	λ 03BB	ŭ 03CB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	Ɔ 038C	M 039C	ά 03AC	μ 03BC	ό 03CC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	N 039D	έ 03AD	ν 03BD	ύ 03CD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E		Υ 038E	Ξ 039E	ή 03AE	ξ 03BE	ώ 03CE
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	– 2015	Ω 038F	Ο 039F	ı̇ 03AF	ο 03BF	

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-8

Latin / Hebrew

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0			א 05D0	ג 05E0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091		± 00B1			ב 05D1	ס 05E1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	¢ 00A2	² 00B2			ג 05D2	ע 05E2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3			ד 05D3	ף 05E3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	´ 00B4			ה 05D4	פ 05E4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	¥ 00A5	µ 00B5			ו 05D5	ץ 05E5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6			ז 05D6	צ 05E6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7			ח 05D7	ק 05E7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8			ט 05D8	ר 05E8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	¹ 00B9			י 05D9	ש 05E9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ª 00AA	º 00BA			ך 05DA	ת 05EA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB			כ 05DB	
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC			ל 05DC	
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD			ם 05DD	LRM 200E
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	¾ 00BE			נ 05DE	RLM 200F
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	— 00AF			– 2017	ן 05DF	

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-9

Latin-5 / Turkish

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 00C0	Ğ 011E	à 00E0	ğ 011F
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	ı 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	´ 00B4	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	¥ 00A5	µ 00B5	Å 00C5	Õ 00D5	å 00E5	ö 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	¦ 00A6	¶ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	¨ 00A8	¸ 00B8	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Í 00CD	İ 0130	í 00ED	ı 0131
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Î 00CE	Ş 015E	î 00EE	ş 015F
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	– 00AF	¿ 00BF	Ĭ 00CF	ß 00DF	ï 00EF	ÿ 00FF

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-10

Latin-6 / Nordic

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	Ā 0100	Đ 00D0	ā 0101	đ 00F0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ą 0104	ą 0105	Á 00C1	Ŋ 0145	á 00E1	ŋ 0146
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	Ě 0112	ě 0113	Â 00C2	Ŏ 014C	â 00E2	ō 014D
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	Ġ 0122	ğ 0123	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	Ī 012A	ī 012B	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	Ĭ 0128	ĩ 0129	Å 00C5	Õ 00D5	å 00E5	ö 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Ɔ 0136	ķ 0137	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	Š 00A7	· 00B7	Ĺ 012E	Ŭ 0168	ĺ 012F	ŭ 0169
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	Ł 013B	ł 013C	Č 010C	Ø 00D8	č 010D	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	Đ 0110	đ 0111	É 00C9	Ȳ 0172	é 00E9	ȳ 0173
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Š 0160	š 0161	Ę 0118	Ú 00DA	ę 0119	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	Ʀ 0166	Ƨ 0167	Ě 00CB	Ű 00DB	ě 00EB	ű 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	Ž 017D	ž 017E	Ě 0116	Ü 00DC	é 0117	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	– 2015	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	Ŭ 016A	ŭ 016B	Î 00CE	Ɔ 00DE	î 00EE	Ɔ 00FE
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Ŋ 014A	ŋ 014B	İ 00CF	ß 00DF	ĩ 00EF	κ 0138

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-11

Latin / Thai

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	ฐ 0E10	ภ 0E20	ะ 0E30	ล 0E40	อ 0E50
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	ก 0E01	ท 0E11	ม 0E21	ั 0E31	แ 0E41	ด 0E51
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	ข 0E02	ฒ 0E12	ย 0E22	า 0E32	โ 0E42	บ 0E52
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	ช 0E03	ณ 0E13	ร 0E23	ำ 0E33	ใ 0E43	น 0E53
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	ค 0E04	ด 0E14	ถ 0E24	ี 0E34	ไ 0E44	จ 0E54
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	ศ 0E05	ต 0E15	ล 0E25	ื 0E35	จ 0E45	ฉ 0E55
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	ฆ 0E06	ถ 0E16	ภ 0E26	ี 0E36	จ 0E46	บ 0E56
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	ง 0E07	ท 0E17	ว 0E27	ี 0E37	ี 0E47	ฌ 0E57
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	จ 0E08	ธ 0E18	ศ 0E28	ุ 0E38	อ 0E48	ฌ 0E58
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	ฉ 0E09	น 0E19	ษ 0E29	ู 0E39	อ 0E49	ฌ 0E59
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ช 0E0A	บ 0E1A	ส 0E2A	ุ 0E3A	อ 0E4A	ฌ 0E5A
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	ช 0E0B	ป 0E1B	ท 0E2B		อ 0E4B	ฌ 0E5B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	ณ 0E0C	ผ 0E1C	พ 0E2C		อ 0E4C	
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	ญ 0E0D	ผ 0E1D	อ 0E2D		อ 0E4D	
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	ฎ 0E0E	พ 0E1E	ษ 0E2E		อ 0E4E	
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	ฏ 0E0F	พ 0E1F	ย 0E2F	ธ 0E3F	อ 0E4F	

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-12

Latin / Devanagari

The work in making a 8859 codepage for Devanagari – an abugida (a segmental writing system based on consonants where vowels are secondary notation) alphabet used in India and Nepal – was never completed and officially abandoned in 1997. (See page 11 of *Understanding Typography*.)

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090						
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091						
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092						
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093						
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094						
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095						
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096						
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097						
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098						
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099						
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A						
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B						
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C						
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D						
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E						
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F						

Identical to ASCII

How Is Type Encoded?

ISO/IEC 8859-13

Latin-7 / Baltic Rim

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 0104	Š 0160	ą 0105	š 0161
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	" 201D	± 00B1	Ĳ 012E	Ń 0143	ĳ 012F	ń 0144
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Ā 0100	Ŧ 0145	ā 0101	ŗ 0146
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ć 0106	Ó 00D3	ć 0107	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	¤ 00A4	“ 201C	Ä 00C4	Õ 014C	ä 00E4	õ 014D
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	„ 201E	µ 00B5	Ħ 00C5	Ö 00D5	ĥ 00E5	ö 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	ı 00A6	ŧ 00B6	Ɛ 0118	Ö 00D6	ę 0119	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ê 0112	× 00D7	ê 0113	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	Ø 00D8	ø 00F8	Č 010C	Ȳ 0172	č 010D	ȳ 0173
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	¹ 00B9	É 00C9	Ł 0141	é 00E9	ł 0142
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Ŕ 0156	ŗ 0157	Ž 0179	Š 015A	ž 017A	ś 015B
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ê 0116	Ȳ 016A	ê 0117	ȳ 016B
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	¼ 00BC	Ğ 0122	Ü 00DC	ğ 0123	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	½ 00BD	Ɔ 0136	Ž 017B	ķ 0137	ž 017C
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	¾ 00BE	Ĭ 012A	Ž 017D	ī 012B	ž 017E
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Æ 00C6	æ 00E6	Ł 013B	ß 00DF	Į 013C	’ 2019

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-14

Latin-8 / Celtic

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	Ĥ 1E1E	À 00C0	Ŵ 0174	à 00E0	ŵ 0175
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ĭ 1E02	ĥ 1E1F	Á 00C1	Ŋ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	ĳ 1E03	Ĝ 0120	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	ġ 0121	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	Č 010A	Ĺ 1E40	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	ć 010B	ŕ 1E41	Å 00C5	Õ 00D5	å 00E5	õ 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Đ 1E0A	Ŧ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	š 00A7	Ř 1E56	Ç 00C7	Ţ 1E6A	ç 00E7	ř 1E6B
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	Ű 1E80	ŵ 1E81	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	þ 1E57	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	Ų 1E82	ŵ 1E83	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	đ 1E0B	Š 1E60	Ě 00CB	Û 00DB	ě 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	Ÿ 1EF2	ÿ 1EF3	ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	SHY 00AD	Ű 1E84	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	ŵ 1E85	Î 00CE	Ÿ 0176	î 00EE	ÿ 0177
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Ÿ 0178	ś 1E61	Ĭ 00CF	ß 00DF	ĩ 00EF	ÿ 00FF

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-15

Latin-9

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 00C0	Đ 00D0	à 00E0	đ 00F0
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	ı 00A1	± 00B1	Á 00C1	Ñ 00D1	á 00E1	ñ 00F1
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	¢ 00A2	² 00B2	Â 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	£ 00A3	³ 00B3	Ã 00C3	Ó 00D3	ã 00E3	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	€ 20AC	Ž 017D	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	¥ 00A5	μ 00B5	Å 00C5	Õ 00D5	å 00E5	ö 00F5
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Š 0160	ŋ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	× 00D7	ç 00E7	÷ 00F7
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	š 0161	ž 017E	È 00C8	Ø 00D8	è 00E8	ø 00F8
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	¹ 00B9	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	ª 00AA	º 00BA	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	¬ 00AC	Œ 0152	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	ŠHY 00AD	œ 0153	Í 00CD	Ý 00DD	í 00ED	ý 00FD
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	® 00AE	Ÿ 0178	Î 00CE	Ɔ 00DE	î 00EE	Ɔ 00FE
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	– 00AF	¿ 00BF	Ĭ 00CF	Ɔ 00DF	ï 00EF	ÿ 00FF

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

ISO/IEC 8859-16

Latin-10 / South-eastern European

Unicode values for each character are listed in gray below the character. These do not necessarily match the hex code for the character on the current table.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070	PAD 0080	DCS 0090	NBSP 00A0	° 00B0	À 00C0	Đ 0110	à 00E0	đ 0111
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071	HOP 0081	PU1 0091	Ą 0104	± 00B1	Á 00C1	Ń 0143	á 00E1	ń 0144
2	STX 0002	DC2 0012	“ 0022	2 0032	B 0042	R 0052	b 0062	r 0072	BPH 0082	PU2 0092	ą 0105	Č 010C	Ā 00C2	Ò 00D2	â 00E2	ò 00F2
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073	NBH 0083	STS 0093	Ł 0141	ı 0142	Ā 0102	Ó 00D3	ă 0103	ó 00F3
4	EOT 0004	DC4 0014	\$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074	IND 0084	CCH 0094	€ 20AC	Ž 017D	Ä 00C4	Ô 00D4	ä 00E4	ô 00F4
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075	NEL 0085	MW 0095	„ 201E	” 201D	Ć 0106	Õ 0150	ć 0107	õ 0151
6	ACK 0006	SYN 0016	& 0026	6 0036	F 0046	V 0056	f 0066	v 0076	SSA 0086	SPA 0096	Š 0160	ŋ 00B6	Æ 00C6	Ö 00D6	æ 00E6	ö 00F6
7	BEL 0007	ETB 0017	‘ 0027	7 0037	G 0047	W 0057	g 0067	w 0077	ESA 0087	EPA 0097	§ 00A7	· 00B7	Ç 00C7	Ś 015A	ç 00E7	ś 015B
8	BS 0008	CAN 0018	(0028	8 0038	H 0048	X 0058	h 0068	x 0078	HTS 0088	SOS 0098	š 0161	ž 017E	È 00C8	Ŭ 0170	è 00E8	ű 0171
9	HT 0009	EM 0019) 0029	9 0039	I 0049	Y 0059	i 0069	y 0079	HTJ 0089	SGCI 0099	© 00A9	č 010D	É 00C9	Ù 00D9	é 00E9	ù 00F9
A	LF 000A	SUB 001A	* 002A	: 003A	J 004A	Z 005A	j 006A	z 007A	VTs 008A	SCI 009A	□ 0218	□ 0219	Ê 00CA	Ú 00DA	ê 00EA	ú 00FA
B	VT 000B	ESC 001B	+ 002B	; 003B	K 004B	[005B	k 006B	{ 007B	PLD 008B	CSI 009B	« 00AB	» 00BB	Ë 00CB	Û 00DB	ë 00EB	û 00FB
C	FF 000C	FS 001C	, 002C	< 003C	L 004C	\ 005C	l 006C	 007C	PLU 008C	ST 009C	Ž 0179	Œ 0152	Ì 00CC	Ü 00DC	ì 00EC	ü 00FC
D	CR 000D	GS 001D	- 002D	= 003D	M 004D] 005D	m 006D	} 007D	RI 008D	OSC 009D	ŠHY 00AD	œ 0153	Í 00CD	Ɛ 0118	í 00ED	ę 0119
E	SO 000E	RS 001E	. 002E	> 003E	N 004E	^ 005E	n 006E	~ 007E	SS2 008E	PM 009E	ž 017A	Ÿ 0178	Î 00CE	□ 021A	î 00EE	□ 021B
F	SI 000F	US 001F	/ 002F	? 003F	O 004F	_ 005F	o 006F	DEL 007F	SS3 008F	APC 009F	Ž 017B	ž 017C	Ĭ 00CF	ß 00DF	ĩ 00EF	ÿ 00FF

Identical to ASCII

Identical to Latin-1

How Is Type Encoded?

Big5

Big5 is a character encoding method used in Taiwan, Hong Kong, and Macau for traditional Chinese characters and BoPoMoFo (a.k.a zhuyin fuhao, BoPoMoFo is the official Taiwanese system for phonetically transcribing Chinese). Big5 is a double byte encoding system capable of encoding thousands of characters; however it is always used in conjunction with a single byte encoding format such as ASCII for encoding non-Chinese characters. The Big5 encoding system is divided into zones, which are dedicated sub-groupings within the greater encoding system.

This format has been superseded by Unicode and is being phased out slowly.

Big5 Encoding Zones

Code Range	Description
0x8140 – 0xA0FE	Reserved for user-defined characters, e.g. 造字
0xA140 – 0xA3BF	Graphical characters, e.g. 圖形碼
0xA3C0 – 0xA3FE	Reserved, <i>not</i> for user-defined characters
0xA440 – 0xC67E	Frequently used characters, e.g. 常用字
0xC6a1 – 0xC8FE	Reserved for user-defined characters
0xC940 – 0xF9D5	Less frequently used characters, e.g. 次常用字
0xF9D6 – 0xFEFE	Reserved for user-defined characters

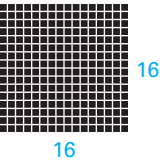
How Is Type Encoded?

Shift JIS

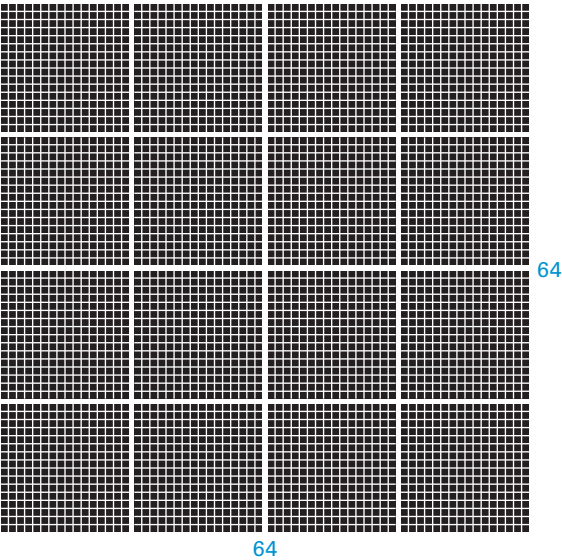
Shift JIS is an encoding system originally developed by the ASCII corporation in conjunction with Microsoft for encoding the Japanese language. Many versions of Shift JIS exist, often with conflicting code points. Because of this, it is recommended that software applications use Unicode instead. Unlike most codepages, which are 16 × 16 in size, Shift JIS is 94 × 94 in size, allowing for a total of 8,836 possible code points (Shift JIS does not fill the entire code space).

This format has been superseded by Unicode and is being phased out slowly.

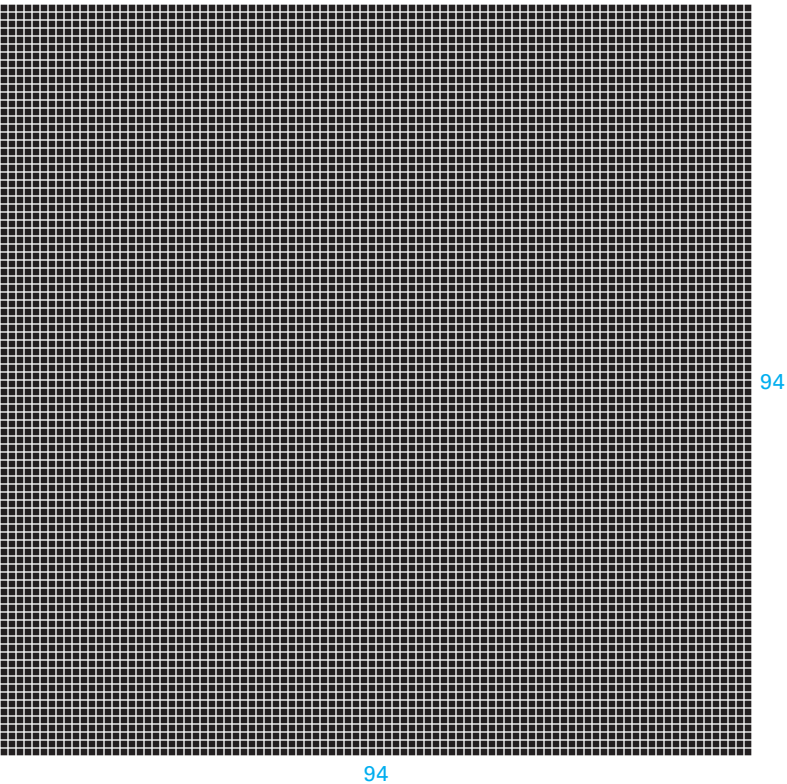
Windows 1252
256 code points
in one codepage



ISO/IEC 8859
256 × 16 (4,096) code points
in 16 codepages



Shift JIS
8,836 code points
in one codepage



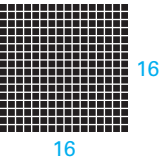
How Is Type Encoded?

KS X 1001

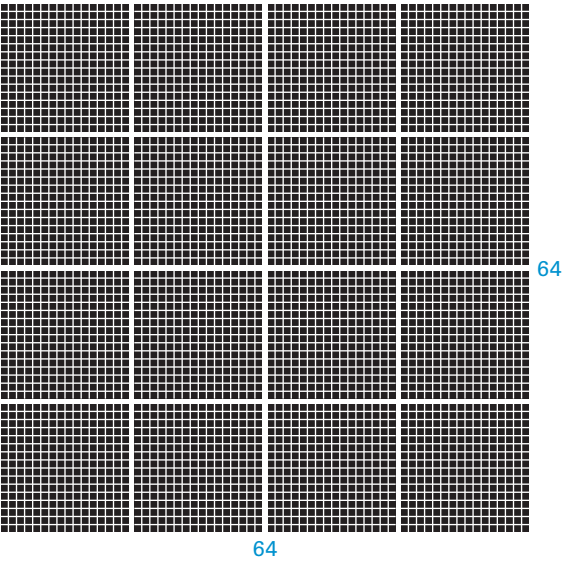
Korean Graphic Character Set for Information Interchange (KS X 1001) is a South Korean character encoding for hangul and hanja ASCII, Greek, Cyrillic, and Japanese kana. KS X 1001 is arranged as 94 × 94 table (similar to Shift JIS).

This format has been superseded by Unicode and is being phased out slowly.

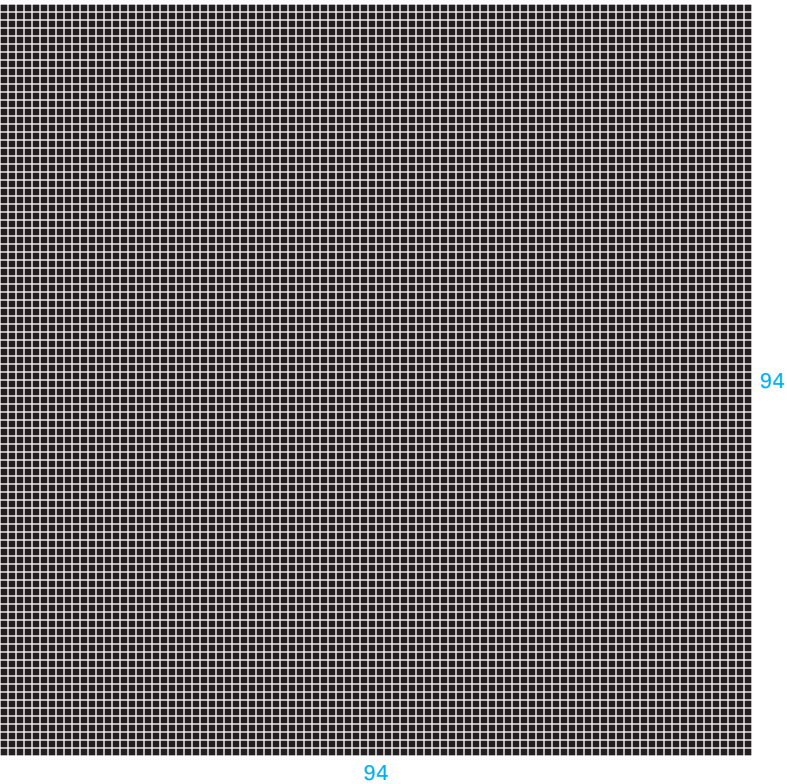
Windows 1252
256 code points
in one codepage



ISO/IEC 8859
256 × 16 (4,096) code points
in 16 codepages



Shift JIS
8,836 code points
in one codepage



How Is Type Encoded?

Unicode

Unicode dates to 1987, when Joe Becker from Xerox and Lee Collins and Mark Davis from Apple started investigating the practicalities of creating a universal character set. The original proposal was for a 16-bit encoding system (65,563 code points) that would be able to handle “all the world’s living languages”. In 1996, the Unicode Consortium, the organization that governs development of the standard, published Unicode 2, which expanded the standard to a 21-bit system so as to no longer be restricted to 16 bits. This added encoding space allowed for the encoding of many historic scripts (e.g. Egyptian Hieroglyphs) and thousands of rarely-used or obsolete characters that had not been anticipated as needing encoding.

Unicode can be implemented by different character encodings.

The most commonly used encodings are UTF-8 (which uses one byte for any ASCII characters and retains the ASCII code values, and up to four bytes for other characters) and UTF-16. The first 256 code points were made identical to the content of ISO 8859-1 so as to make it trivial to convert existing western text.

The current version of Unicode is divided into 17 “planes” – groups of code points designated for specific types of characters. Each plane has 65,536 (= 2¹⁶) code points. This results in a total of 1,114,112 code points*. Currently, about ten percent of the potential space is used. Ranges of characters have been tentatively mapped out for every current and ancient writing system the Unicode consortium has been able to identify. While Unicode may eventually need to use another of the spare 11 planes for ideographic characters, other planes remain, if previously unknown scripts with tens of thousands of characters are discovered. This 21-bit limit is therefore unlikely to be reached in the near future.

* The system to handle this many code points is slightly awkward because a 20-bit (2²⁰) system could handle only 1,048,576 code points, so the designers had to move to a 21-bit system of encoding that has the potential to handle 2,097,152 code points. There is no discussion of whether there any plans to make use of the greater possible capacity of a 21-bit system.

Plane 0 covers 65,536 code points, many more than ISO 8859, which could have covered 2,176 but actually covered far fewer (under 1,000) because there were many repeating characters. All of ISO 8859’s covered characters are also covered by Unicode. Unlike the way in which all of the ISO 8859 parts were backwards compatible with ASCII, the only portion of Unicode’s mapping that matches ISO 8859 is the first 256 characters, which match ISO 8859-1, the rest have been re-mapped.

All of ISO 8859 would fit in this cluster

Unicode Planes & Code Point Ranges

Basic	Supplementary				
0000–FFFF	10000–1FFFF	20000–2FFFF	30000–DFFFF	E0000–EFFFF	F0000–10FFFF
Plane 0: Basic Multilingual (BMP)	Plane 1: Supplementary Multilingual (SMP)	Plane 2: Supplementary Ideographic (SIP)	Planes 3–13: Unassigned (–)	Plane 14: Supplementary Special Purpose (SSP)	Plane 15–16: Private Use Area (PUA)
0000–0FFF	10000–10FFF	20000–20FFF	–	E0000–E0FFF	15: PUA-A
1000–1FFF	11000–11FFF	21000–21FFF			F0000–FFFFF
2000–2FFF	12000–12FFF	22000–22FFF			
3000–3FFF	13000–13FFF	23000–23FFF			16: PUA-B
4000–4FFF		24000–24FFF			100000–10FFFF
5000–5FFF		25000–25FFF			
6000–6FFF	16000–16FFF	26000–26FFF			
7000–7FFF		27000–27FFF			
8000–8FFF		28000–28FFF			
9000–9FFF		29000–29FFF			
A000–AFFF		2A000–2AFFF			
B000–BFFF	1B000–1BFFF	2B000–2BFFF			
C000–CFFF					
D000–DFFF	1D000–1DFFF				
E000–EFFF					
F000–FFFF	1F000–1FFFF	2F000–2FFFF			

The entirety of Unicode’s character encoding map can be found online at: www.unicode.org/charts/

The Unicode consortium’s credo is:
*Unicode provides a unique number for every character,
no matter what the platform,
no matter what the program,
no matter what the language.*

How Is Type Encoded?

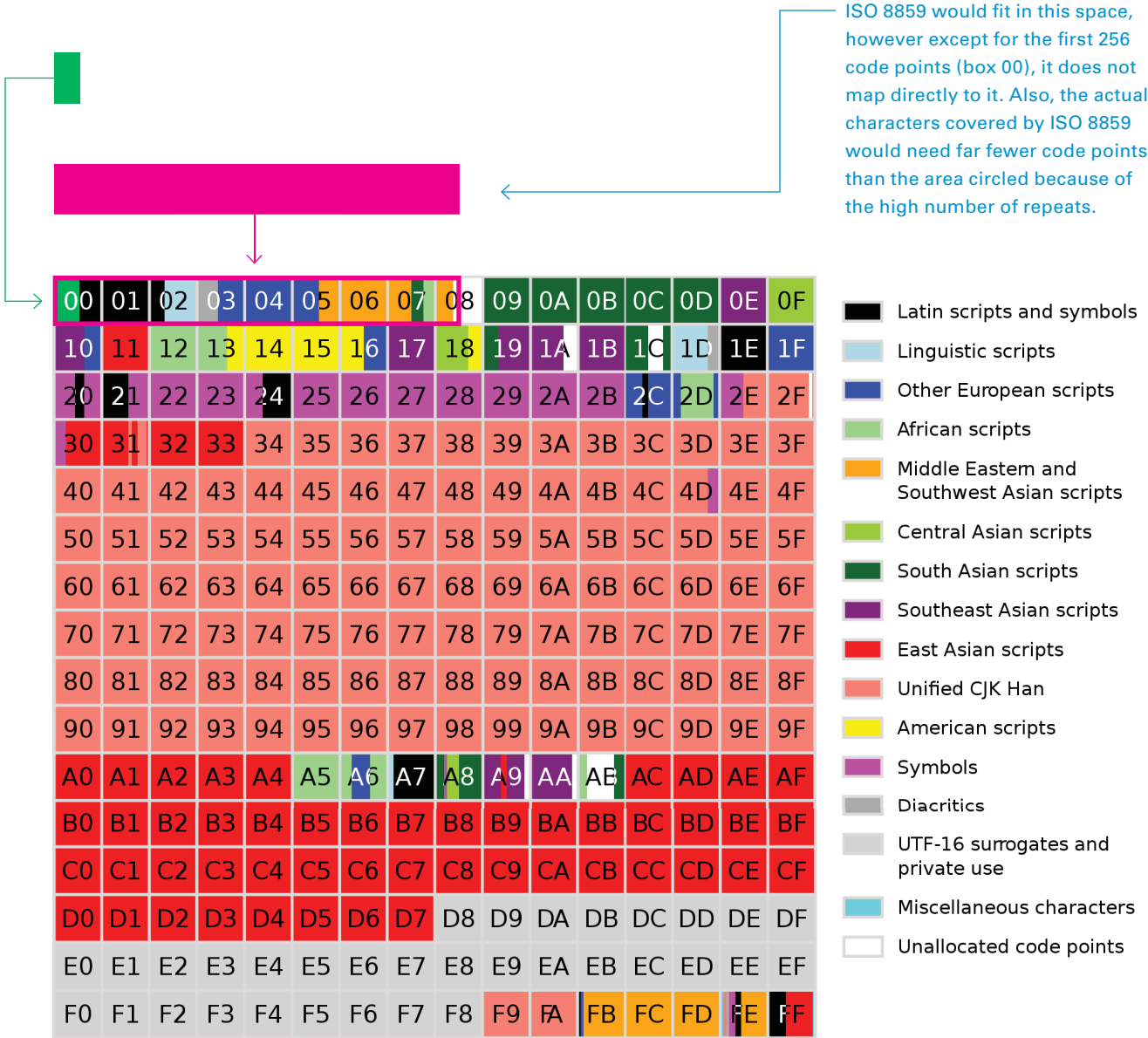
Unicode: Basic Multilingual Plane

The first Unicode plane (Plane 0), the Basic Multilingual Plane (BMP), is where most characters have been assigned. The BMP contains characters for almost all modern languages, and a large number of special characters. A primary objective for the BMP is to support the unification of prior character sets as well as characters for writing. Most of the allocated code points in the BMP are used to encode Chinese, Japanese, and Korean (CJK) characters.

ASCII
(128)

ISO 8859
(2,176 possible)

Basic Multilingual Plane
Plane 0
Each numbered box
represents 256 code points
(65,536)



How Is Type Encoded?

Unicode: 8 vs 16 vs 32

Unlike the previously discussed character encoding systems, which can be only implemented in one way, Unicode code points can be mapped to “code values” (sequences of values, e.g. 00410A) in several ways. There are several versions of the Unicode Transformation Format (UTF), most notably UTF-8, UTF-16, and UTF-32.

While these three versions of UTF can all represent every character in the Unicode character set, they perform their encoding in different ways. UTF-8 is the only one of these versions of UTF that is backward-compatible with ASCII. The other primary difference between the versions is that while UTF-8 and UTF-16 are variable width encoding, UTF-32 is fixed width encoding. Variable width encoding uses fewer bytes for lower numerical values. Fixed width encoding uses the same number of bytes for all code points. Variable width encoding is more efficient than fixed width encoding. For these and other reasons, UTF-8 has become the dominant character encoding for the World-Wide Web, accounting for more than half of all webpages.

In webpages, the character encoding is indicated in the header of the file:

```
<Content-Type: text/html; charset=utf-8>
```

ASCII

ISO 8859
(up to 000880)

Basic Multilingual Plane (Plane 0)

Code Range (hexadecimal)	UTF-8	UTF-16	UTF-32
000000 – 00007F	1 byte	2 bytes	4 bytes
000080 – 00009F	2 bytes		
0000A0 – 0003FF			
000400 – 0007FF	3 bytes		
000800 – 003FFF			
004000 – 00FFFF			
010000 – 03FFFF	4 bytes	4 bytes	
040000 – 10FFFF			

One curiosity is that while UTF-8 encodes points 000600 – 00FFFF in 3 bytes, this is not necessary. It is unclear why this would be done.

How Is Type Encoded?

Unicode: Fonts

Just because the Unicode standard exists does not mean all fonts make use of its possibilities. Today most typefaces still only work with a 256 character set. The barriers to taking full advantage of the Unicode standard are mainly time: designing a typeface, in a single weight and style, for the Latin-1 set can take months, possibly years. Creating the glyph drawings for the full Unicode Plane 0 is a massive undertaking.

Sources:
www.creativepro.com/blog/typetalk-character-reference

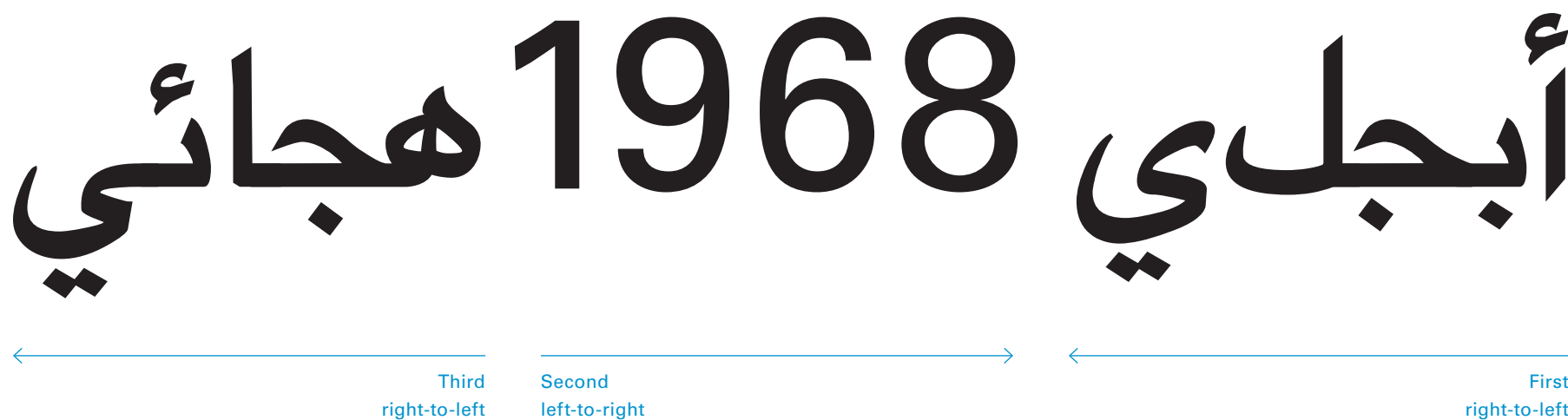
Multi-language “Super Fonts”

Name	Characters (#)	Glyphs (#)	File Size	Font Format
Arial	3,415	3,415	756 KB	OTF + TTO
Arial Unicode MS	38,917	50,377	22.10 MB	OTF + TTO
Bitstream Cyberbit	32,961	50,377	12.70 MB	TTF
Bitstream Cyber CJK	30,275	28,686	12.40 MB	TTF
Caslon Roman	3,683	3,686	3.70 MB	TTF
Code2000 *	53,068	63,546	7.98 MB	TTF
DejaVu Sans	5,467	5,762	667 KB	OTF + TTO
FreeSerif	7,203	8,995	1.60 MB	TTF
GNU Unifont	63,446	63,449	15.5 MB	Bitmap, TTF
HAN NOM A	32,328	34,147	20.30 MB	TTF
Lucida Grande	2,245	2,826	1.07 MB	OTF
Microsoft Sans Serif	2,788	3,077	637 KB	OTF + TTO
New Gulim	46,567	49,284	24.50 MB	TTF
Tahoma	1,912	3,412	681 KB	OTF + TTO
Times New Roman	2,790	3,414	816 KB	OTF + TTO
TITUS Cyberbit Basic	9,209	10,044	1.91 MB	TTF
WenQuanYi Bitmap Song	41,295	154,997	-	Multi-strike bitmap
WenQuanYi Zen Hei	42,285	43,643	16.00 MB	TTC
WenQuanYi Micro Hei	34,707	48,755	-	TTC
Y.OzFontN	21,957	57,621	13.50 MB	TTC

* While Code2000 has many characters, the quality of Latin & Hangul is poor.

Unicode: Bi-direction

Early computer systems were designed to support only one writing direction, typically left-to-right. Adding new character encodings enabled a number of new left-to-right scripts to be supported, but did little to aid in the use of right-to-left scripts. Beyond providing a way to encode more than a million characters, **Unicode is important because it added support for bi-directional scripts: text containing both directions in a single line.** Unicode accomplishes this by assigning a direction and strength to all characters. All areas of the code map have a default direction assigned to them, e.g. all code points that fall within the Arabic section of Plane 0 by default will run from right-to-left. All characters also have a strength. “Strong” characters, such as letters, have a direction that they always adhere to. “Weak” characters, such as numerals and punctuation, have a direction that they use when on their own, however when they are inside a string of strong characters they rely on the direction of those strong characters.



A single line of editable text can have multiple reading directions. The string above (أبجدي [ʾabġadī], هجائي [hiġāʾī]) reads right-to-left but the numbers are read left-to-right (“1968”) as compared to right-to-left (“8691”).

How Is Type Encoded?

GB18030

GB18030 is a Chinese government encoding standard published in 2000. Prior to GB18030, the primary encoding standard used was GBK (Guojia Biao zhun Extension), which was developed in 1996 and encoded 21,886 characters. GBK was an extension of GB2312, which encoded 7,445 characters. GBK incorporated the Unified Han portion of Unicode 2.1 that goes beyond the character repertoire of GB2312. GB18030 extends GBK further, incorporating the remainder of the Unicode 3.0 code space.

GB18030 is not perfectly compatible with Unicode however, and does not map directly to it. GB18030 can cover a total of 1,587,600 code points (more than Unicode’s 1,114,112), leaving about 500,000 byte sequences in GB18030 unassigned.

Comparison of GB 18030 to Unicode Versions UTF-8, UTF-16, UTF-32

Code Range (hexadecimal)	GB18030	UTF-8	UTF-16	UTF-32
000000 – 00007F	1 byte	1 byte	2 bytes	4 bytes
000080 – 00009F	2 bytes	2 bytes		
0000A0 – 0003FF	Characters encoded here are inherited from GB2312 / GBK	2 bytes		
000400 – 0007FF				
000800 – 003FFF		3 bytes		
004000 – 00FFFF				
010000 – 03FFFF	4 bytes	4 bytes	4 bytes	
040000 – 10FFFF				

How Are Character Shapes Represented?

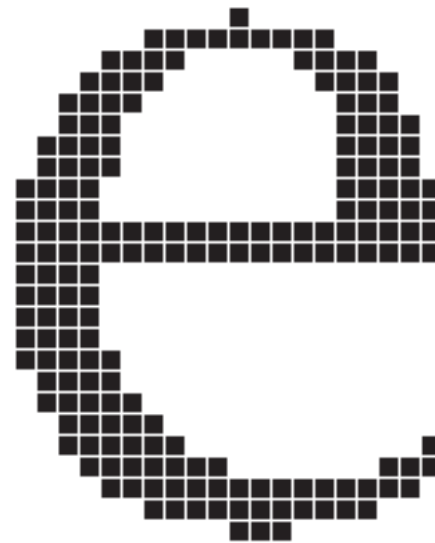
The technology for representing a character has evolved over time from symbols hand-drawn with a stylus, brush, or pen to characters pressed onto paper by solid blocks of metal bearing raised, reversed characters to digital coordinate outlines resized, transformed, and set by mathematical formulas. Type design for metal required the designer to conceive not only of the overall look of the typeface but each size-specific version with its own optical tweaks and variations. Type design was about creating a unified set of sets. Digital type can be any size and modern technology means it can be reproduced in a bewildering number of ways – printed on paper, fabric, or plastic; displayed on all sorts of computer screens; standing still or animated and morphed in three dimensions.

How Are Character Shapes Represented?

Bitmap Fonts

Bitmap or raster fonts were designed to work within the constraints of low-resolution computer monitors. Bitmap fonts, like metal fonts, had to be designed for specific point sizes. For example, a bitmap font file would typically contain 9, 12, 18, and 24 pixels-per-em (PPEm*) bitmap sets. If a user tried to use the font at a size that was not part of the pre-designed set, the rendering engine would pick the nearest size and reduce it to fit. At small sizes this often led to very poor results.

Today bitmap fonts are less common on PCs due to higher resolution monitors and greater processing power. However there are still many devices with simple display technology that still make use of bitmap fonts.



* Computer display resolutions are measured in pixels-per-inch (PPI). In the 80s, Microsoft set the default PPI for their operating system at 96 PPI. Apple set the default PPI for their computers at 72 PPI to corresponded with the traditional typographic standard of 72 points in an inch. Type set at 72 PPI on screen should be exactly the same size as 72 point type on paper. While this was useful in 1984, screen resolutions today are typically much higher than 72 PPI. Pixels-per-em (see page 59 of *Understanding Typography* for more information on ems), or PPEm, is the height of a bitmap character's bounding box in pixels. How big a font of a given PPEm appears on a screen depends on the physical size of the pixels themselves. For example, a 24 PPEm glyph displayed on a 1984 Macintosh monitor would be 24 points tall (about 1/3 inch). The same glyph displayed on a 96 PPI Windows monitor would be 18 points tall (about 1/4 inch). Each glyph would be 24 pixels high – it's just that the Mac monitor pixels are taller.

A bitmap glyph. The design of a bitmap font is very crude when compared to type design for metal or photo-setting. The font is composed of square units that are either on or off. There are no partial units.

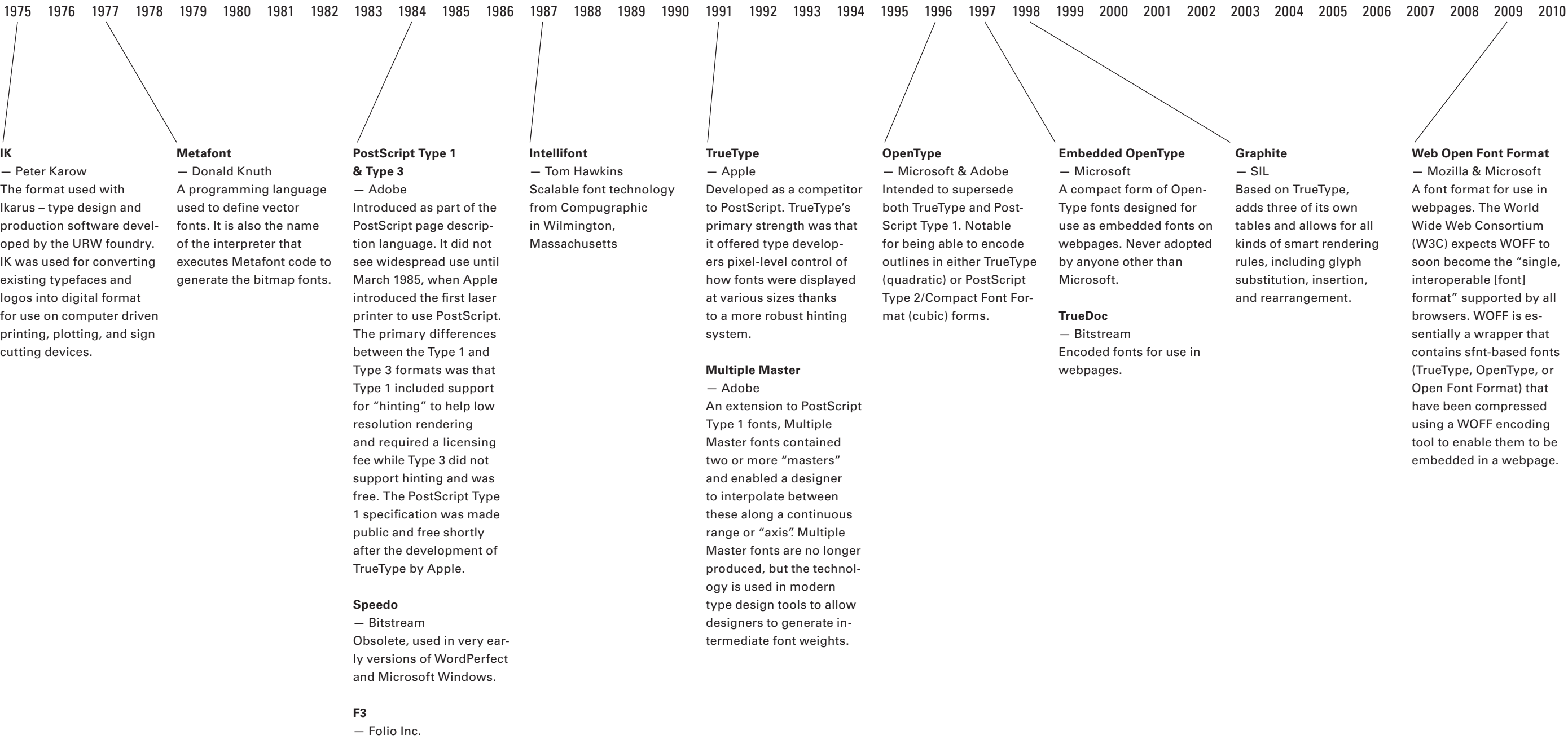


The typeface Chicago, as seen on the third generation iPod. Chicago was originally designed as a bitmap font for the Macintosh, but it was redrawn as an outline font when the Macintosh computer had enough processing power to render screen bitmaps from outlines.

How Are Character Shapes Represented?

Digital Font File Formats

An overview of the major digital font file formats.



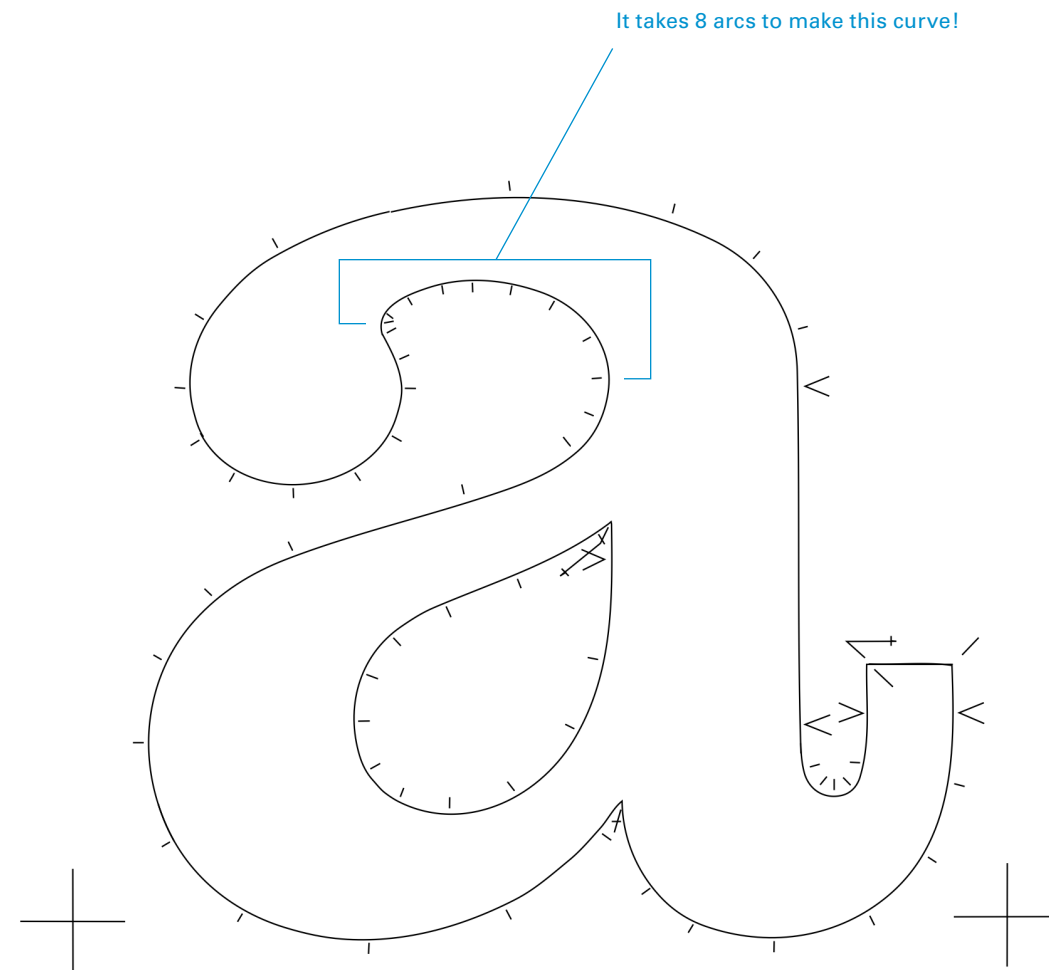
How Are Character Shapes Represented?

Ikarus

Designed in 1975 by Peter Karow while working at the URW type foundry, Ikarus is a type design and production software tool for converting existing typefaces and logos into digital format for use on computer driven printing, plotting, and sign cutting devices. Ikarus uses a spline model of outline description, storing the outline as a set of coordinates and arc radii. This abstract mathematical approach allowed for a perfectly scalable description of each glyph. The curve segments are all circle arcs, with tangent continuity at the joints. Because of the nature of curves in Ikarus, complex curve shapes required many segments.

Trivia fact #1: The original work for Ikarus came from programs for designing ship's hulls.

Trivia fact #2: Ikarus got its name because of the frequency with which it crashed in the early days of development, likening it to the Greek character of Icarus ("Ikarus" is the German spelling).



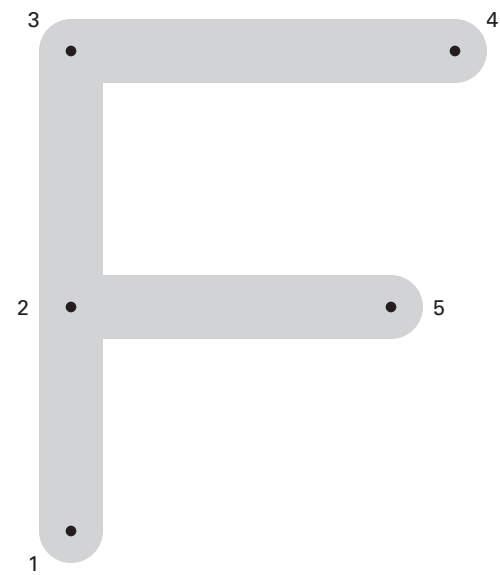
A lower-case "a" drawn in Ikarus. The small tick marks demarcate the edges of curves; the < symbols mark tangent points, and the larger ticks mark corners. Note how many arcs are required to compose some of the curves on the shape.

How Are Character Shapes Represented?

Metafont

Designed by Donald Knuth in 1977, Metafont is a typeface design and description language. Instead of storing glyphs as sculpted objects (metal) or images (phototype), Metafont stores glyphs as a set of numbers that can be interpreted by a set of equations. Unlike outline font descriptions, Metafont works by defining a stroke (or strokes) with finite-width “pens”, along with filled regions. Thus, instead of describing the glyph as a filled outline that is the image of the glyph, Metafont describes pen paths. Some simple Metafont fonts (e.g. Computer Modern) use a single pen stroke with a relatively large pen to define each stroke of a glyph. Other Metafont fonts use two paths, the space between which would be filled (this working in a similar way to an outline font).

Since the font shapes are defined by equations rather than directly-coded numbers, it is possible to treat parameters such as aspect ratio, font slant, stroke width, serif size, and so forth as input parameters in each glyph definition (which then define not a single font, but a *meta*-font). Thus, by changing the value of one of these parameters at one location in the Metafont file, one can produce a consistent change throughout the entire font.



A simple Metafont font and the code used to produce it:

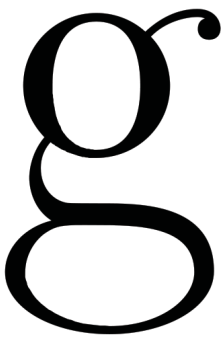
```
1 beginlogochar("F", 14);
2 x1 = x2 = x3 = leftstemloc;
3 x4 = w - x1 + ho;
4 x5 = x4 - xgap;
5 y2 = y5;
6 y3 = y4;
7 bot y1 = -o;
8 top y3 = h;
9 y2 = barheight;
10 draw z1
11   -- z3
12   -- z4;
13 draw z2
14   -- z5;
15 labels(1, 2, 3, 4, 5);
16 endchar;
```

What this means:

```
1 Make an "F", it is glyph 14
2 1, 2, & 3 are aligned horizontally,
3 4 is to the right of 1
4 5 is to the left of 4
5 5 is level with 2
6 4 is level with 3
7 1 is at the bottom
8 3 is at the top
9 2 is in the middle
10 Draw a line from 1
11   (through 2) to 3
12   then on to 4
13 Draw a line from 2
14   to 5
15 Label the points
16 Stop, the character is drawn
```

What isn't included here is what the height, width, mid-point, and "gap" dimensions are. This would have been defined in a global table used for the entire font.

The Quick Brown
Fox Jumps Over
The Lazy Dog.
abcdefghijklmnopqrstuvwxyz0123456789 [] () { } /\ < >



A more complex Metafont font. The definition for this font would have 2 paths and a fill between them. Unlike an outline font though, the weight of the font can be altered easily by changing the stroke thickness (this will also result in the terminals being rounder).

How Are Character Shapes Represented?

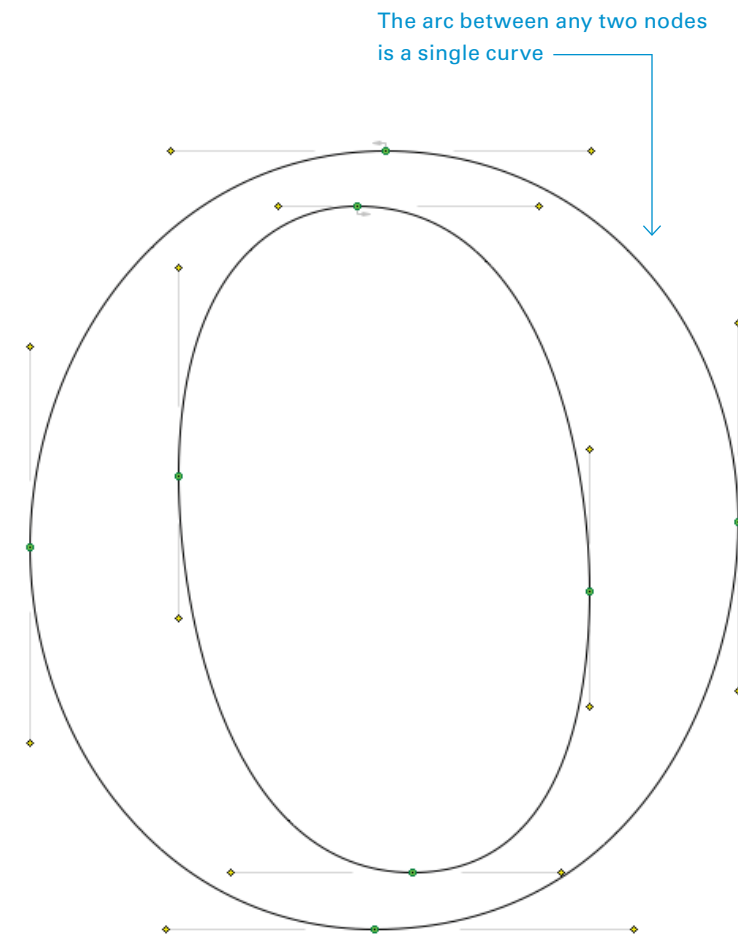
PostScript & TrueType

Contemporary font files almost all use one of two outline formats: **PostScript or TrueType**. This naming scheme is confusing because PostScript and TrueType are the names of both outline formats (PostScript cubic curves and TrueType quadratic curves) and font file formats (PostScript Type 1 fonts and TrueType fonts), which are not the same thing. Modern font formats such as OpenType and Web Open Font Format (WOFF) still use either PostScript or TrueType outline formats at their core. However, while PostScript Type 1 fonts can only use PostScript outlines, and TrueType fonts can only use TrueType outlines, OpenType fonts can use either PostScript or TrueType outlines, but OpenType cannot mix outline formats in a single font file.

At the most basic level, **the difference between the two outline formats is purely mathematical: PostScript uses cubic Bézier curves while TrueType uses quadratic B-splines**. Quadratics are simpler than cubics, and for simple shapes should require fewer points. For example, a circle described with cubic curves requires 12 points while the same circle as described by quadratic curves should only need 8 (though the author has never been able to produce one in practice). For more complex shapes – and glyphs are rarely simple – **cubic curves typically have fewer points**.

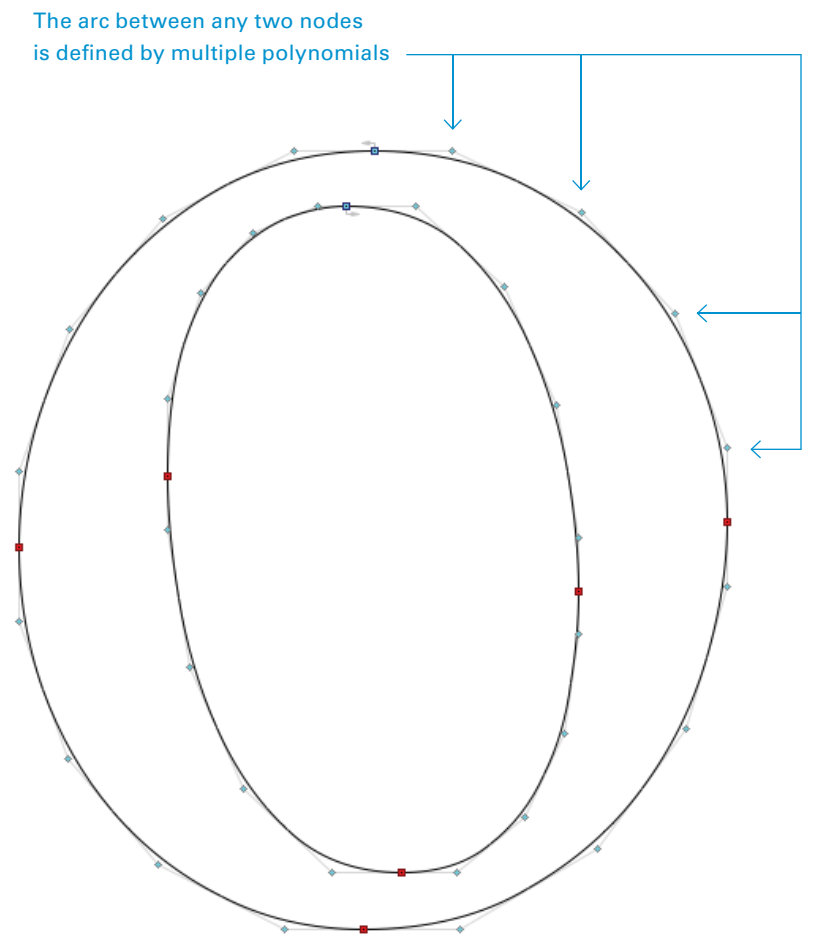
PostScript outlines consist of “nodes” and “control handles”, with the control handles being used to manipulate the curve between any two adjacent nodes. Each node can have a maximum of two handles. TrueType has “on-curve” and “off-curve” points. Between any two on-curve points there may be many off-curve points. Both formats can be scaled in an infinitely smooth and precise manner. This works very well with high resolution output devices such as printers. However, when resolution is limited, for example, on a computer monitor, rendering problems can arise.

Converting between the two formats is a lossy process due to rounding errors. The errors are greater when converting from PostScript to TrueType.



PostScript Uses Cubic Curves

When drawing a font using these curves, the user places “nodes” (the green dots) that define the general outline, and then uses “control handles” to define the curve (if any) between the nodes. Each node can have two handles, one in each direction. In the example above, there are a total of eight nodes (four for the outer shape and four for the inner shape), each of which have two control handles, for a total of 24 points (or coordinates). If the lines defined by a node and its two handles are continuous, the connection is smooth. Otherwise, the connection is “sharp”, and the contours may change abruptly at the node.



TrueType Uses Quadratic Curves

Drawing a quadratic curve is in some ways similar to drawing a cubic curve. However the way they are controlled is very different. This stems from the fact that while everything between nodes on a cubic curve is actually one curve mathematically, the line defined between two on-curve points (the red and larger blue squares above) of a quadratic curve is usually composed of two or more polynomials. These polynomials connect to form the curve, and individually they are mathematically simpler than a cubic bezier curve. Unfortunately, to construct a shape of any complexity, you need many polynomial segments (in the example above, there are three or four polynomials between each pair of on-curve points). This makes drawing a font outline tedious. First, the type designer must manipulate more points for any given curve. Second, not all polynomial points are connected directly to on-curve points: manipulating curves visually by hand often results in lumpy shapes.

Sources:

www.true-type-typography.com/articles/ttvst1.htm

blogs.adobe.com/typblography/2010/12/the-benefits-of-opentypecff-over-truetype.html

developer.apple.com/fonts/TTRefMan/RM01/Chap1.html

How Are Character Shapes Represented?

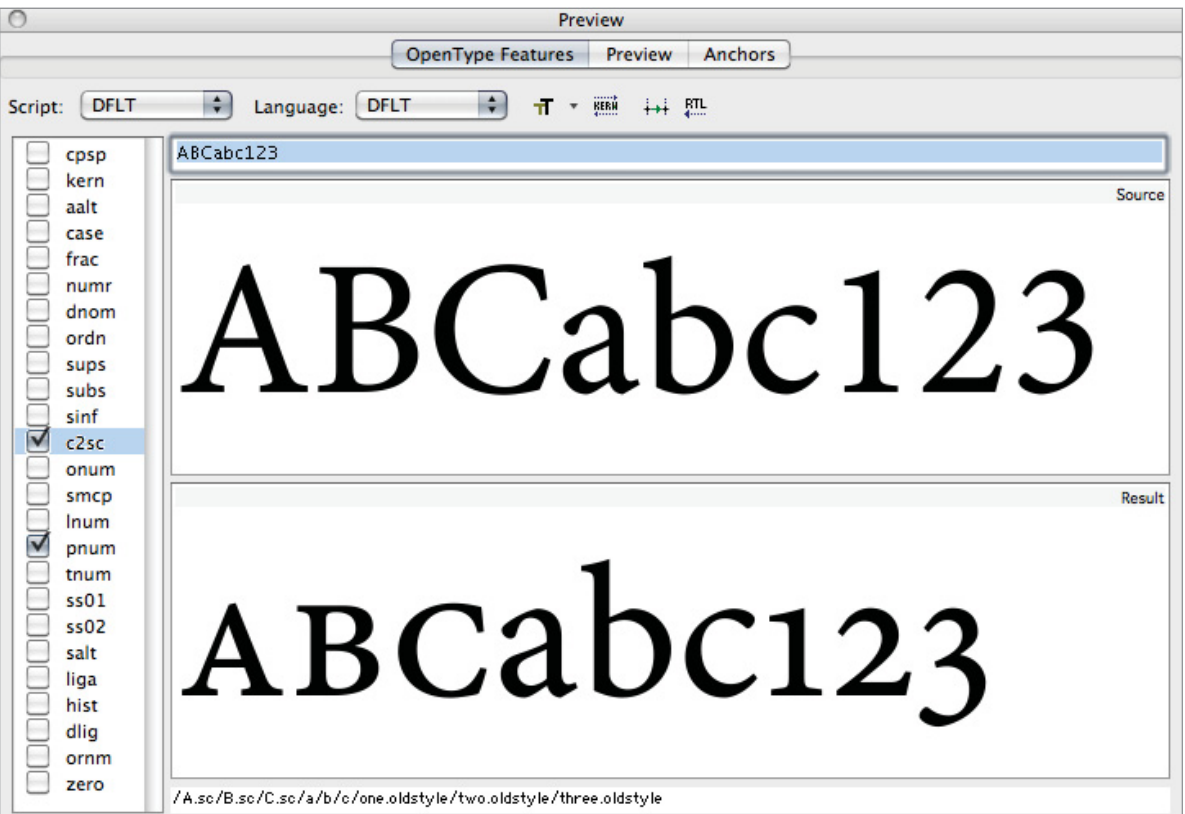
OpenType

OpenType was developed in 1996 by Microsoft and Adobe with the intention that it would **replace both PostScript and TrueType** font formats. The primary intention of OpenType was to provide robust **cross-platform support for more writing systems**. The format allows for far **greater control of contextual shaping** and “smart” typography, such as allowing **a single font file to contain both lining and old style numerals as well as small caps**, features that would have required separate files for each in PostScript and TrueType formats. This has been especially important for typography using non-alphabetic writing systems (e.g. abjad, abugida, etc.). OpenType also supports Unicode, and any OpenType font can have up to 65,536 glyphs – the same number as a single Unicode plane. Unlike PostScript and TrueType, OpenType can use either cubic or quadratic curves (but only one at a time).

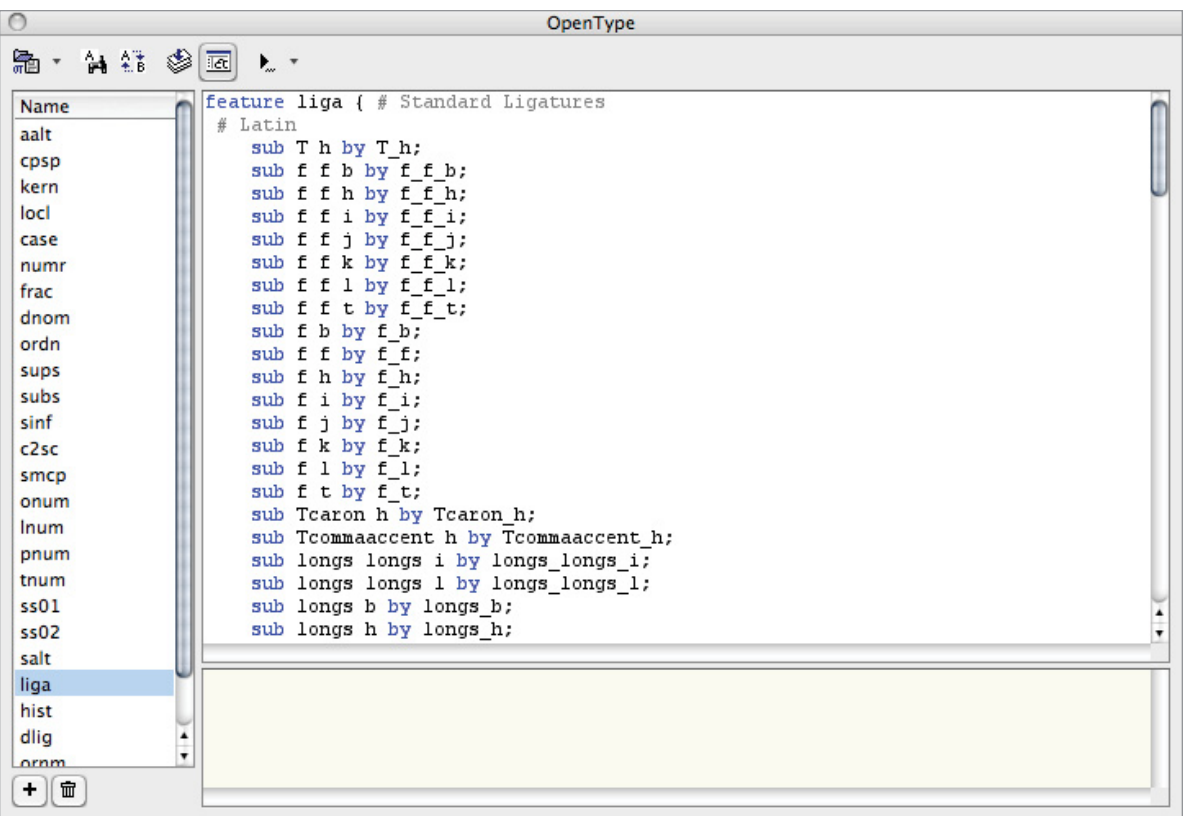
Open Font Format, 2009, is an open source specification identical to OpenType 1.4.

Aside from better support for more language scripts, OpenType’s main advancement is the power to define layout features. These features can be context-dependent substitutions such as ligatures, or they can be selectively activated by the user. Programs like Adobe InDesign support selective features turned on through menu commands. Examples of selective features include switching to small caps or alternate numeral forms by clicking a button. Prior to OpenType, setting small caps or alternate numeral forms would have required the user to select an entirely different font file containing the lesser used characters. With OpenType, they can all be included in the same font file.

The example to the right shows the change from the standard glyphs to an alternate set (“c2sc” stands for “caps to small caps”, and “onum” stands for “old style numerals”).



OpenType layout features are defined in code language developed by Adobe called the OpenType Feature Definition Language. It is considered the best way of defining layout features. The example to the right shows how the code is written for defining ligatures.



How Are Character Shapes Represented?

Contemporary Font Format Comparison

Not all font file formats allow for both types of curves. PostScript font files can only have PostScript cubic curves, while TrueType can only have quadratic curves. Modern font file formats can have either cubic or quadratic curves, but they cannot have both in the same font file.

On average OpenType font files are 20% – 50% smaller than comparable TrueType fonts due to:

– **File storage**

OpenType relies on “subroutinization”, a process that surveys all the glyphs in the font looking for path segments that are identical. Identical path segments are replaced by a shared routine, which reduces the amount of data in the file. TrueType has a similar process, but it is not as effective as OpenType’s.

– **Hinting data**

OpenType font files have much less hinting data than TrueType font files, again reducing the file size. When it comes to hinting, the two formats have very different approaches. OpenType fonts prefer to rely on the intelligence of the renderer more than TrueType, which prefers to be as explicit as possible with hinting data and treats the renderer as a “dumb” machine to carry out its very exacting instructions. As rendering engines get better, OpenType’s method may win out because it will require far less effort on the part of the designer to achieve satisfactory results. Additionally, as rendering engines improve, OpenType fonts improve with them, while TrueType fonts do not!

File size differences have implications for webfonts, and they also have huge ramifications for CJK (Chinese, Japanese, Korean) fonts which contain tens of thousands of glyphs and files ranging around 5 – 10 MB (or more!).

Sources:
blog.typekit.com/2010/12/08/type-rendering-font-outlines-and-file-formats/
blogs.adobe.com/typblography/2010/12/the-benefits-of-opentypecff-over-truetype.html

Font File Formats	Outline Formats		Notes
	Cubic (PostScript)	Quadratic (TrueType)	
PostScript Type 1 (.pfm)	●		Superseded by OpenType
TrueType (.ttf)		●	
OpenType (.otf)	●	●	Only one format per font file
Embedded OpenType (.eot)	●	●	
Web Open Font Format (.woff)	●	●	

How Do Computers Display Type?

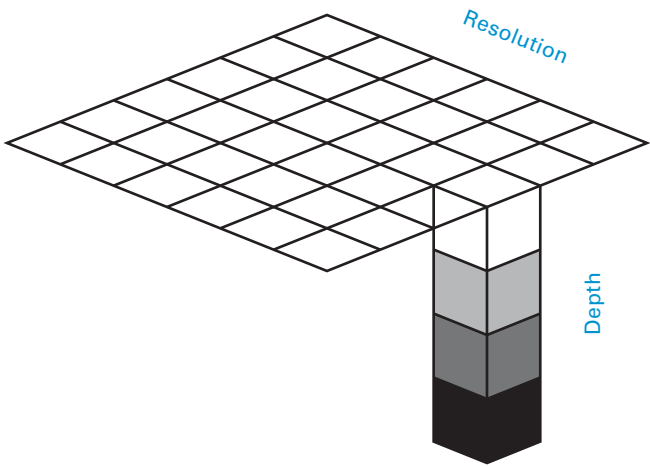
Font rasterization is the process of converting a glyph outline description (as found in scalable fonts such as TrueType) to a raster or bitmap description and displaying it on screen. Rasterization usually takes place at the OS level and often involves some anti-aliasing of bitmaps to make them smoother and easier to read on screen. It may also involve hinting, that is, the use of pre-drawn pixel images for a particular font size to improve the appearance of the bitmap.

How Do Computers Display Type?

Bit Depth

The number of bits used to represent the color of a single pixel in a bitmapped image is called bit depth. At the low end, a depth of 1-bit means that a pixel can be either black or white (1-bit = 2 colors). As bit depth increases to 2-bits, the number of possible colors for each pixel is raised to a power of 2 (2-bit = $2^2 = 4$ colors). Each additional bit increases the number of supported colors by a factor of two. For example, a bit depth of 4 = $2^4 = 16$ colors; a bit depth of 8 = $2^8 = 256$ colors. 1-bit color depth is always monochromatic and typically black and white. 2-bit depth is usually black and white but may be color. 4-bit and beyond is typically color. The number of bits per pixel relates to image quality because a greater bit depth means there are more possible colors and shades available for any given pixel, allowing the image to be rendered with greater subtlety.

Higher bit depth color spaces require more processing power from the computer, so the history of bit depth evolution closely mirrors the evolution of processing power in computer chips.



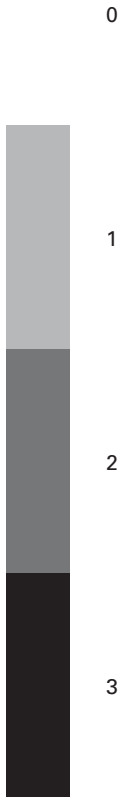
Bit depth and resolution are independent factors that affect image quality. High bit depth is not the only way to achieve high quality images – is it possible to have a high quality image with low bit depth if the resolution is high enough. Often there is a trade-off to be made between high bit depth and high resolution.

How a smooth gray ramp is rendered in:

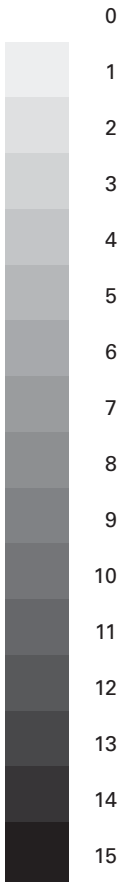
1-bit
 $2^1 = 2$ grays



2-bit
 $2^2 = 4$ grays



4-bit
 $2^4 = 16$ grays

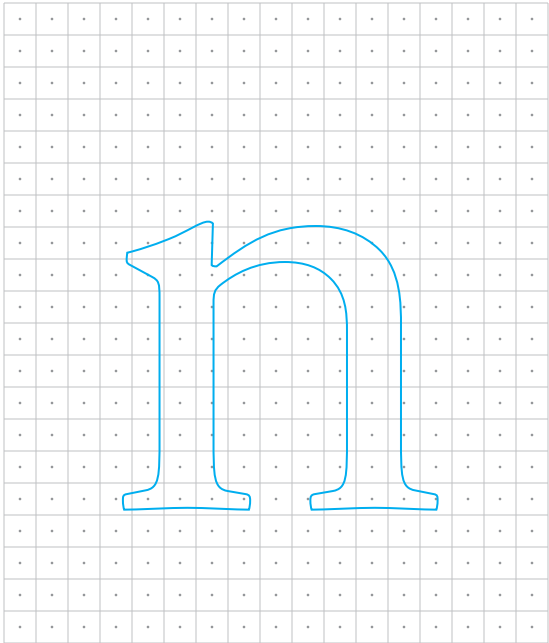


8-bit
 $2^8 = 256$ grays

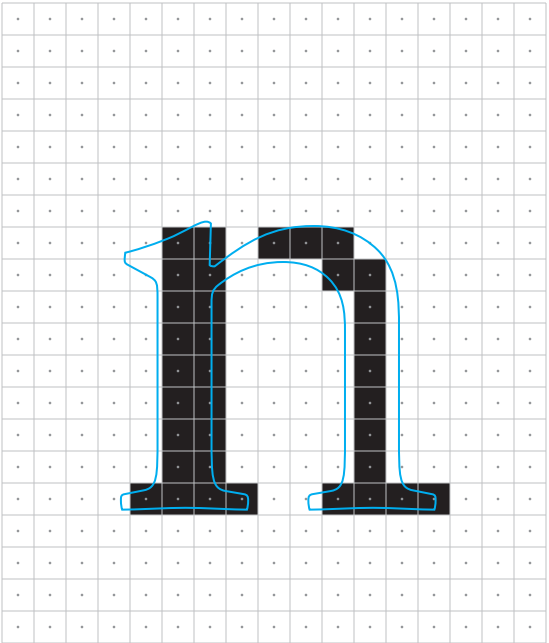


Outline vs Pixel

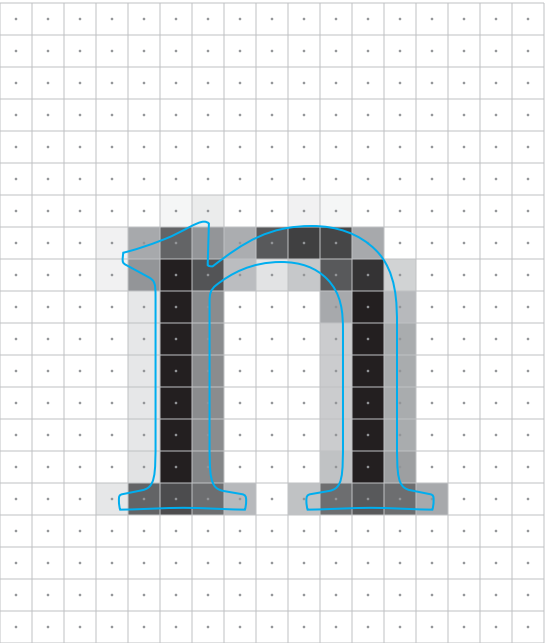
The outlines that describe glyph shapes have to be rasterized into pixels before they can be displayed on screen. The simplest way to render a glyph is to turn on any pixel whose center falls within the glyph outline. However, if the glyph outline inadvertently encloses too many – or too few – pixel centers the resulting onscreen character can be anything from ugly to unreadable. Missing pixels can pose as much of a problem as extra pixels.



Outline On the Grid
Notice that the outlines do not match up neatly with the grid.



Black and White Pixel Conversion
Creating a pixel shape based on whether the center of each grid square was inside or outside of the outline produces a strange form with some parts missing and others inconsistent.



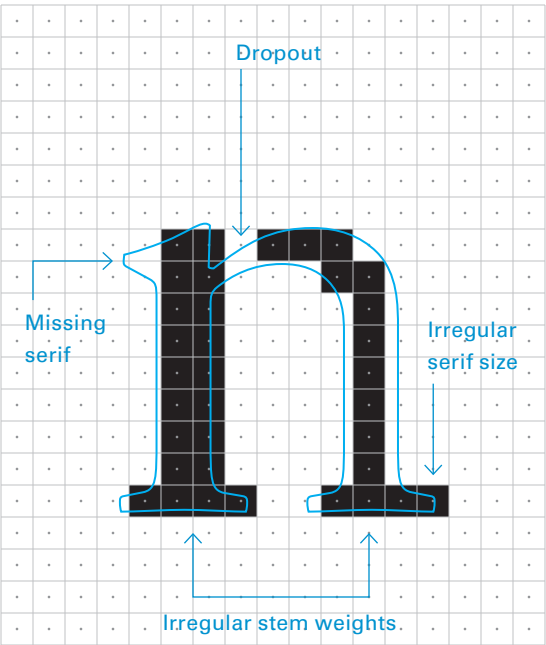
Grayscale Pixel Conversion
Instead of pixels being on or off, this method determines the tone of each pixel based on how much of each pixel is inside or outside of the outline. The resulting form is less strange but appears blurry and might be hard to decipher at small sizes.

Hinting

Font hinting (also known as instructing) is the use of additional instructions to adjust the display of an outline font so that glyphs look better at small sizes. It was first introduced in 1984 as part of the PostScript Type 1 font format as a way to reconcile the way data was stored (mathematical outlines) with the way the data would be output (pixels on screen, or dots from a printer). Several years later, TrueType would extend the power of hinting greatly, making it possible for designers to specify the appearance of glyphs at any and every size (and then some), but also making font design a far more time consuming process.

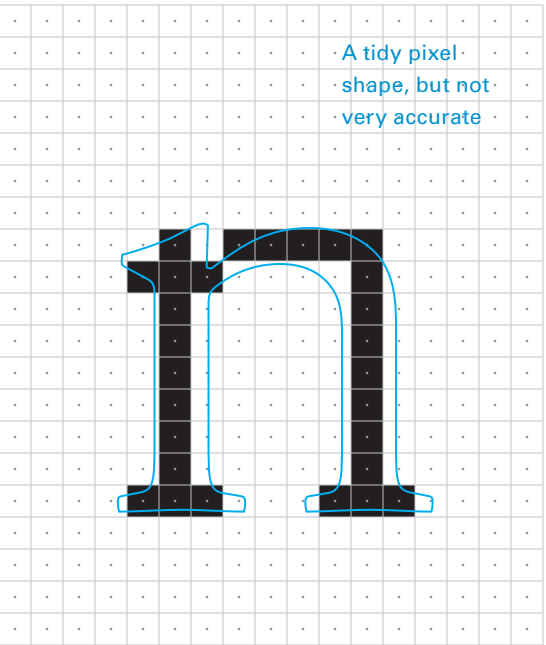
A font can be hinted either automatically (through processed algorithms based on the character outlines) or set manually. Most fonts are automatically hinted by the font drawing program (e.g. FontLab). Automatic hinting is handled by an engine inside the type design program that creates hints for all glyphs at a set of default sizes based on simple, generic rules designed to work well for as many typefaces as possible. While automatic hinting is sufficient for many fonts, manually hinting is typical of higher quality typefaces because it usually takes into account special cases and problematic situations such as “dropout” cases, in which there are no full pixels within range of the outline and the rasterizer would render nothing, thus breaking the form of the glyph.

PPEm is an abbreviation for “pixels per em” – it is the unit of measurement type designers use when hinting to determine how many pixels will make up the em square. (See page 59 of *Understanding Typography* for more information on ems.) It is more accurate than giving a point size, because point size may vary between monitors that have different pixel resolutions. PPEm is resolution independent. A font hinted at 20 PPEm will have the same hints regardless of whether the monitor displaying the image is 72 dpi or 300 dpi – even though the displayed image will be very different in measurable size, it will be made of the same number of pixels.



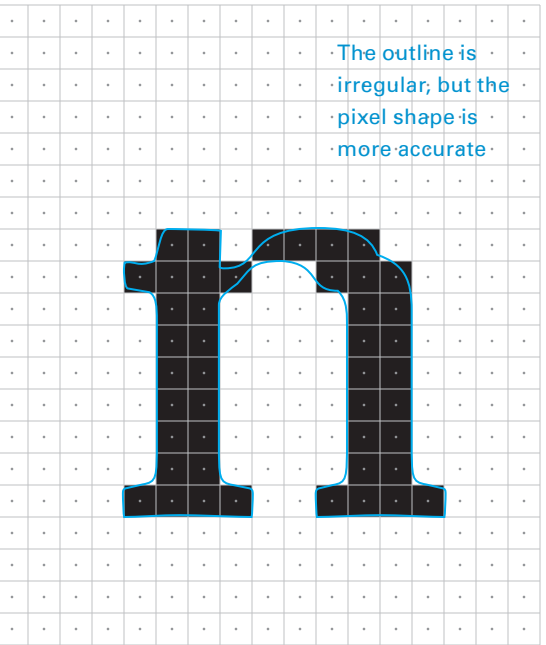
Unhinted (at 20 PPEm)

Without any hinting, the pixel shape created from a font outline can bear little resemblance to the designer's intention. The most common problems with unhinted fonts are dropouts, irregular stem weights, and colliding pixels. Dropouts, or missing pixels, happen where parts of the font outline do not contain pixel centers. Irregular weights occur when the outline shape does not fall neatly onto the grid and some stems, which are identical in the outline drawing, become different thicknesses in the pixel shape. Colliding pixels occur when two parts of the outline shape that are not supposed to touch generate overlapping pixels due to the coarseness of the pixel grid. This problem can be severe when rendering complex characters (e.g. Chinese) with too few pixels.



Base Hinted (at 20 PPEm)

Base hinting is used to preserve and regularize stems, features, and spacing at all sizes. The hinter (either a designer or engineer) can “link” stems to make sure that they are always the same weight. The hinter can also specify that parts of a glyph must always connect, thus preventing dropout. The example form above is now regularized and tidy, however it doesn't match the outline very well.



Delta Hinted (at 20 PPEm)

Delta hints are size specific, and nudge a point to turn a pixel on or off. Delta hints go beyond base hinting to try to make the character shape at any given size more visually faithful to the designers intention. Notice how the outline shape is deformed but the pixel shape more closely resembles the original outline when compared to the previous illustration.

Aliasing

When a computer digitizes a signal – like turning an analog form (e.g. the curve of a letter) into a digital form (e.g. a font bitmap), it can introduce spurious information. The technical term for this process is “aliasing”.

In older systems, type was rendered in the simplest manner: pixels being either completely on or off with no intermediate gray values. Text displayed in this way was made from pre-drawn bitmaps of specific sizes. This method is very fast as it requires the fewest computational cycles to render type on screen. Outline font technology promised to allow any size font to be drawn on screen, not just fixed, hand-drawn sizes. However, outline-generated bitmaps, especially at smaller sizes, were often illegible. Adobe PostScript changed this by allowing designers to include hints with the outline font file to make sure the rendered, aliased font aligned well to the screen grid.

The word "sample" is rendered in a pixelated, blocky font style. The letters are composed of discrete black pixels on a white background, with no smooth curves or gradients. This results in a jagged, stepped appearance, particularly noticeable in the vertical strokes and the curves of the 'a' and 'p'.

Type rendered from an outline font file with no hinting shows the problem of aliasing.

The word "sample" is rendered in the same pixelated font style as above, but with improved alignment to the screen grid. The letters are more uniform in width and height, and the curves are better approximated by the pixel grid, resulting in a more legible and consistent appearance.

Type rendered from an outline font file with hinting still shows aliasing, but the letterforms are better aligned to the screen grid.

Anti-Aliasing

Anti-aliasing reduces the effects of aliasing. Originally the technique was developed by engineers at MIT and Xerox, who called it half-bitting. They recognized that the appearance of low-resolution screen fonts could be improved by taking advantage of computer screens' ability to display intermediate gray values between black and white. Anti-aliasing works by determining how much of any given pixel is covered by the outline glyph to be rendered and then drawing that pixel with that percentage of black. This technique can produce blurry glyphs at small sizes. For instance, if a vertical stroke is supposed to be one pixel wide, but falls half-way between two screen pixels, the result would be a 2 pixel-wide 50% gray line. Hinting is used for anti-aliasing, as for aliasing, to create pixel shapes that better fit the screen grid. However with anti-aliasing the resulting curves and diagonals appear smoother through the use of gray tones.

The image shows the word "sample" in a lowercase, sans-serif font. The letters are rendered with a pixelated, blurry appearance. The edges of the characters are not sharp, and the interior of the letters is filled with a mix of black and gray pixels, giving it a soft, out-of-focus look. This is the result of anti-aliasing without hinting.

Type rendered from an outline font with no hinting – anti-aliasing improves the appearance compared to aliasing alone.

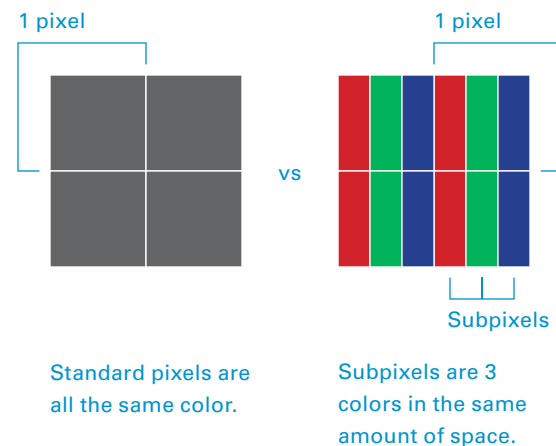
The image shows the word "sample" in the same lowercase, sans-serif font. The letters are rendered with a pixelated, sharp appearance. The edges of the characters are more defined, and the interior of the letters is filled with a mix of black and gray pixels, giving it a sharper, more defined look. This is the result of anti-aliasing with hinting.

Type rendered from an outline font with hinting and anti-aliasing.

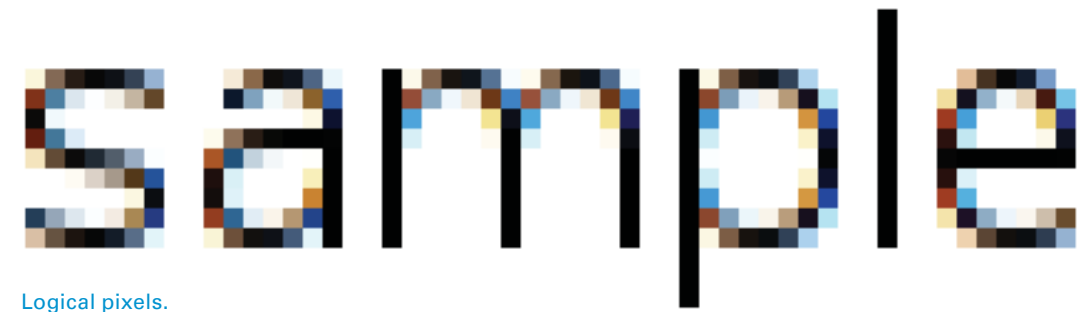
How Do Computers Display Type?

Subpixel Rendering

LCD screens have square pixels that are each comprised of three “subpixels” – red, green, and blue vertical stripes. Because of the way the human visual system blurs and interpolates small visual elements, red, green, and blue light sources will combine to appear white if they are small enough. Further, below a certain size threshold, the human eye cannot perceive the color of a light source but can perceive intensity. Subpixel rendering exploits these characteristics of human vision to create single color letterforms that appear to be drawn at a higher resolution than the rest of the display. Subpixel rendering works best with black on white letters, but will work reasonably well with any solid color. Multicolored shapes do not work well at all and are often rendered with standard anti-aliasing.



- Subpixel rendering engines are included in:
- ClearType (Microsoft)
 - DirectWrite (Microsoft)
 - Mac OS X Quartz
 - RISC OS
 - Adobe CoolType (PDFs)
 - FreeType
 - D-Type Font Engine



Logical pixels.



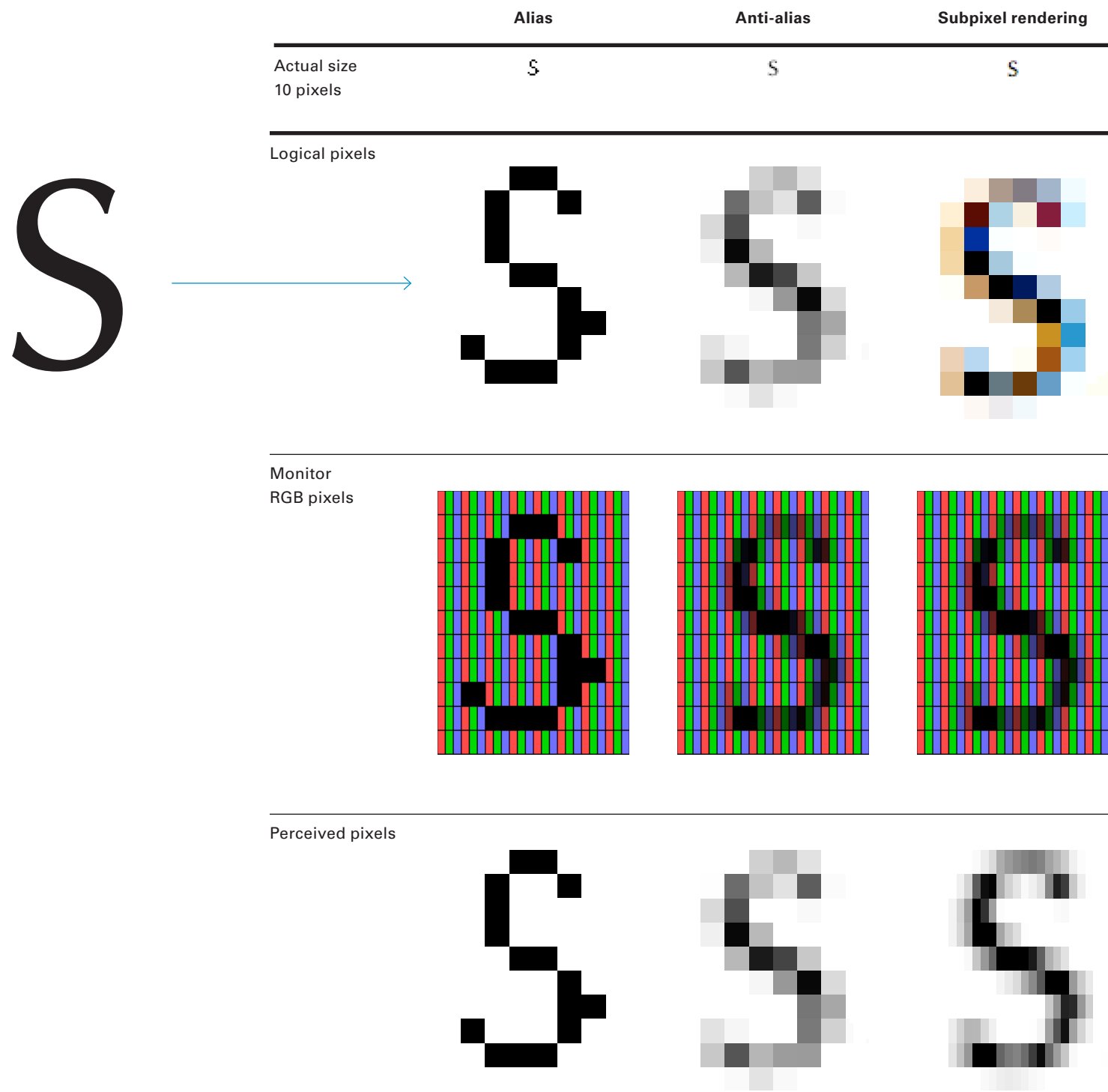
Subpixels on an LCD screen. The word here would appear white on black.



Perceived pixels.

Comparing Rendering Strategies

There are three basic onscreen rendering strategies: aliasing, anti-aliasing, and subpixel rendering. Aliased text is very sharp at small sizes but often does not create a very accurate representation of an outline font. Anti-aliased type corrects for some of the accuracy issues of aliased type but doesn't go as far as subpixel rendering. Subpixel rendering, however, only works on LCD and OLED (organic light emitting diode) displays, which have thin, vertical red, green, and blue light emitters. Subpixel rendering creates the most accurate shape in relation to the outline font but may result in significant color fringing – the appearance of colored pixels at the edges of letters – an effect caused by the values of subpixels.



Rendering Engines Comparison

Aliasing

Graphics Device Interface (GDI)
Windows 7, Vista, and XP

Rare. Almost all modern computers have the processing power to anti-alias text. The user can intentionally choose to display aliased text or a setting in the font file disables anti-aliasing for a given size.

No gray tones,
pixels are either on or off



five boxing
my gods j

abcefghijop 12

The five boxing wizards j
Brawny gods just flocke
Waltz, bad nymph, for qu

Anti-aliasing

Graphics Device Interface (GDI)
Windows 7, Vista, and XP

Windows Standard anti-aliasing is in grayscale only. This is how most Windows XP users see type on the Web (unless they browse with Internet Explorer 7 or 8 because ClearType was turned on by default in those browsers) and how most users with CRT monitors see type as well.



five boxing
my gods j

abcefghijop 12

The five boxing wizards j
Brawny gods just flocke
Waltz, bad nymph, for qu

Subpixel Rendering

Graphics Device Interface (GDI)
with ClearType enabled
Windows 7, Vista, and XP

ClearType does not anti-alias in the y-direction. Note the abrupt, staircase-like curves and color fringing in the capital "R" detail. The latest version of ClearType may not be present in a given Web browsing experience.

Lack of y-direction anti-aliasing,
jagged "stair-like" curve
Color fringing



five boxing
my gods j

abcefghijop 12

The five boxing wizards j
Brawny gods just flocke
Waltz, bad nymph, for qu

DirectWrite
with ClearType enabled
Windows 7 and Vista

DirectWrite has less intense color fringing than the Mac OS and older Windows text rendering engines. There is y-direction anti-aliasing for smooth curves. DirectWrite pays more attention to the pixel grid and cleans up edges when possible.

Y-direction
anti-aliasing



five boxing
my gods j

abcefghijop 12

The five boxing wizards j
Brawny gods just flocke
Waltz, bad nymph, for qu

Core Graphics (Quartz 2D)
Mac OS X and iOS

The Mac OS rendering engine tends to respect a typeface's designed outlines as much as possible. This can make letters, as well as spaces within and among letters, seem blurry at small sizes. However, at very large sizes Core Graphics type looks natural and smooth. This is achieved with y-direction anti-aliasing, which is especially noticeable in the subtle curves of large letters.



five boxing
my gods j

abcefghijop 12

The five boxing wizards j
Brawny gods just flocke
Waltz, bad nymph, for qu

How Do Computers Display Type?

Windows vs Mac

Windows’ ClearType rasterizer tries to align characters to whole pixels vertically and sub-pixels horizontally. The Mac OS’ rasterizer tries to preserve the design of the typeface as much as possible, sometimes at the cost of legibility.

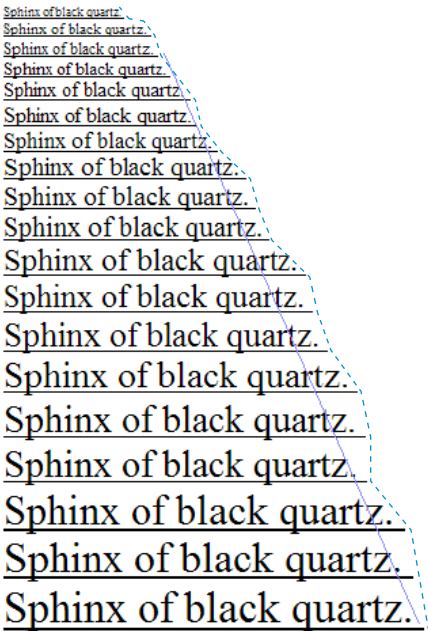
Windows fonts look sharper on screen at the cost of looking less like the original font outline, especially at small sizes. Mac OS fonts look more like the original font outline with some loss of legibility at smaller sizes. Mac OS fonts tend to scale more smoothly than Windows fonts.

Windows’ rasterizing software produces extremely good results with a few built-in TrueType fonts, but sub-optimal results with 99% of other typefaces because most fonts are not hinted correctly, extensively enough, or even at all. The Mac OS Core Text technology ignores font hinting completely and renders all fonts equally well regardless of their font format. Some argue that the Windows approach is better because, through hinting, it produces a sharper image. On the other hand, some advocate for the Mac approach with its “font-smoothing” algorithm because it produces more predictable results and a more accurate rendering in reference to the outline drawing.

Sources:
www.typotheque.com/articles/hinting
www.codinghorror.com/blog/archives/000885.html
www.atpm.com/12.01/paradigm.shtml

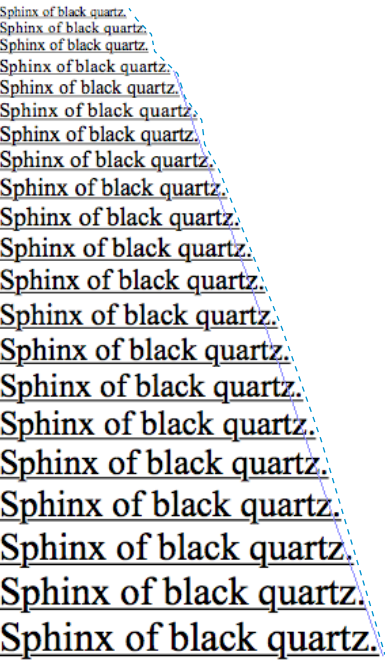
Scaling

Windows



Less smooth scaling.
Pixel-fitting is emphasized.

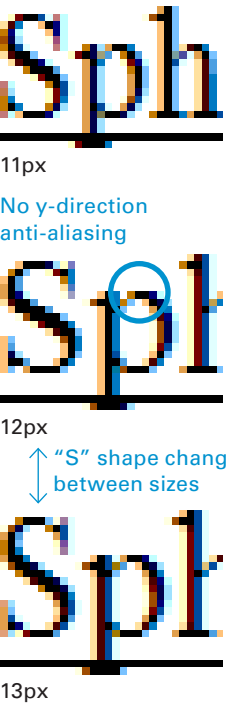
Mac OS



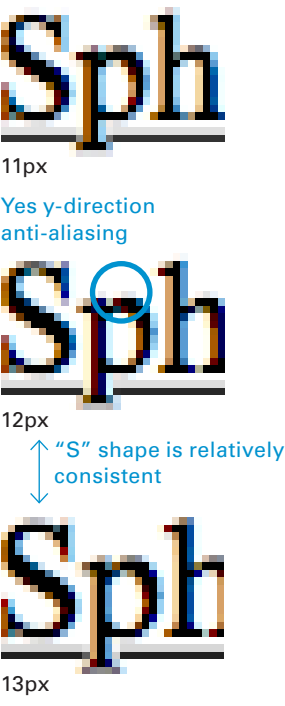
Much smoother scaling, but not perfect,
especially at small sizes.
Outline preservation is emphasized.

Glyph Shape Variety

Windows



Mac OS



Note the variation in glyph shapes on Windows as compared to the relative consistency of the Mac OS results. The Windows approach is consistently sharper due to the lack of y-axis anti-aliasing and emphasis on pixel-fitting. The Mac OS produces heavier weight glyphs due to the disregarding of hinting data.

How Do Computers Display Type?

OS vs Browser

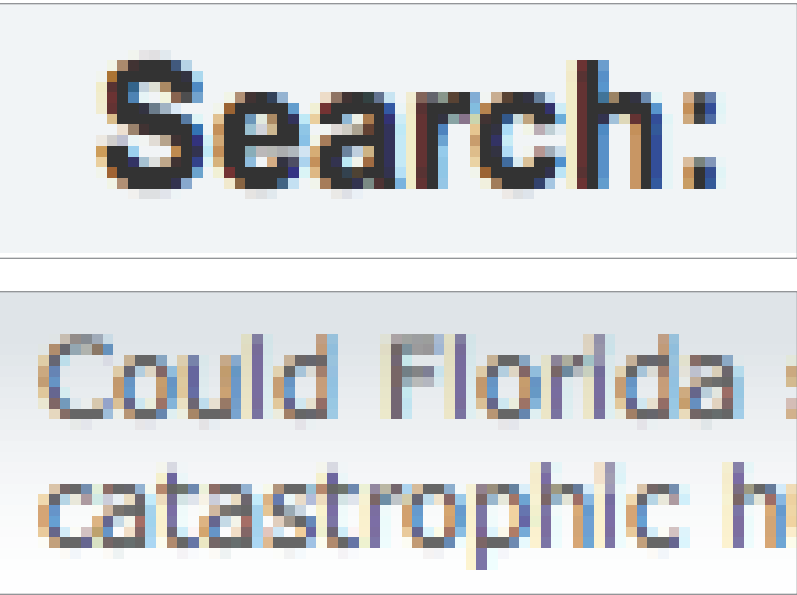
Each operating system has at least one font rendering engine. Each browser decides which rendering engine (if there is more than one) and default settings to use. Therefore on the same OS, two browsers can produce text with very different appearances because they use different rendering engines. On top of that, rendering engines may differ between different versions of the OS and different versions of the browser may use different rendering engines. Lastly, default font-smoothing settings vary by OS and version, and can be overridden by users’ browser preferences.

Operating System	Rendering Engine	
Mac OS X	Core Graphics (Quartz 2D)	
Windows	Graphics Device Interface (GDI)*	* ClearType optional
	DirectWrite*	
Linux	FreeType	

Look closely at the examples of the word “Search”. On Windows Vista, Safari and Internet Explorer both support subpixel anti-aliasing; Firefox and Chrome do not. Only Safari supports subpixel anti-aliasing in the y-direction. Safari for Windows may have its own built-in font rendering engine. Most browsers have their own layout engine but rely on core OS rendering engine to display glyphs. No other browser is known to have its own renderer.

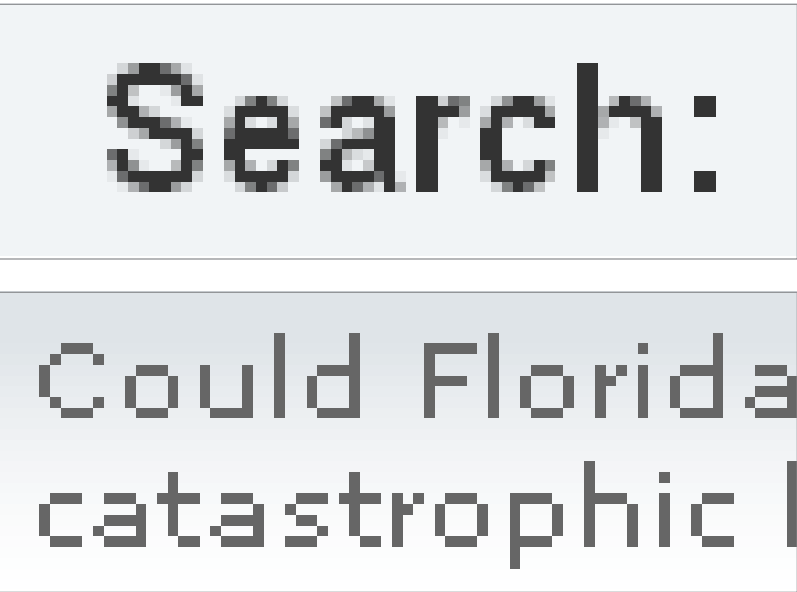
Illustrations are 400% of actual size.

Safari Running on Windows Vista



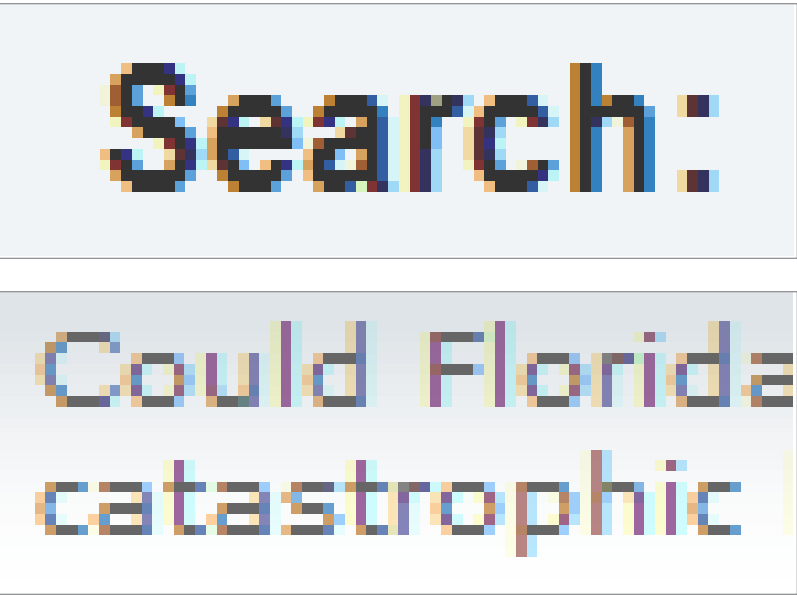
Subpixel rendering is on. Differences in font rendering between Safari and Internet Explorer (IE) are most apparent at small type sizes, where glyph shapes in Safari and IE are extreme. At larger sizes, the most noticeable difference between the two browsers is that Safari has y-direction anti-aliasing.

Firefox Running on Windows Vista



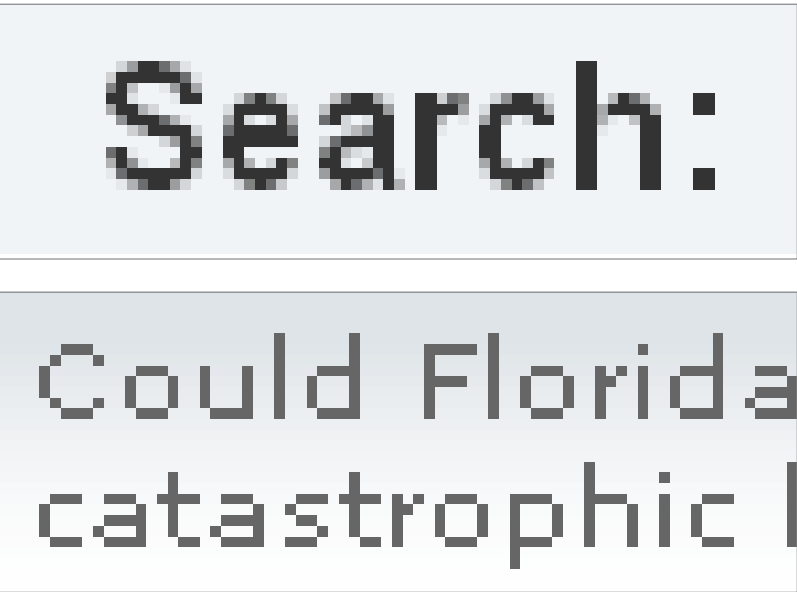
Anti-aliasing is used for larger text while smaller text is aliased.

Internet Explorer (IE) Running on Windows Vista



Subpixel rendering is on, however the results are markedly different when compared to Safari. Letters look lighter in weight, and often show divergence from the character outline shapes to neatly match the pixel grid. Note: Older versions of IE turn ClearType off when certain CSS attributes are activated, most notably the “filter” attribute. This is often used to set transparency settings in IE because it does not conform to the CSS standard.

Chrome Running on Windows Vista



Anti-aliasing is used for larger text while smaller text is aliased. The text here is pixel-by-pixel identical to Firefox.

How Are Fonts Managed?

How Are Fonts Managed?

System Fonts

Fonts that ship with an operating system are referred to as system fonts. Most users are not aware that system fonts are technically not free, as they were licensed by the maker of the operating system prior to launch. Unlike fonts that come packaged with applications or fonts bought from third-party vendors, system fonts cannot be deactivated.

Mac OS 10.5 System Fonts

AmericanTypewriter.dfont	ChalkboardBold.ttf	Hiragino Kaku Gothic ProN W3.otf	Tahoma Bold.ttf
Andale Mono.ttf	Cochin.dfont	Hiragino Kaku Gothic ProN W6.otf	Tahoma.ttf
Apple Braille Outline 6 Dot.ttf	Comic Sans MS Bold.ttf	Hiragino Kaku Gothic Std W8.otf	Thonburi.ttf
Apple Braille Outline 8 Dot.ttf	Comic Sans MS.ttf	Hiragino Kaku Gothic StdN W8.otf	ThonburiBold.ttf
Apple Braille Pinpoint 6 Dot.ttf	Copperplate.dfont	Hiragino Maru Gothic Pro W4.otf	Times LT MM
Apple Braille Pinpoint 8 Dot.ttf	Courier New Bold Italic.ttf	Hiragino Maru Gothic ProN W4.otf	Times New Roman Bold Italic.ttf
Apple Braille.ttf	Courier New Bold.ttf	Hiragino Mincho Pro W3.otf	Times New Roman Bold.ttf
Apple Chancery.dfont	Courier New Italic.ttf	Hiragino Mincho Pro W6.otf	Times New Roman Italic.ttf
Apple LiGothic Medium.dfont	Courier New.ttf	Hiragino Mincho ProN W3.otf	Times New Roman.ttf
Apple Symbols.ttf	Courier.dfont	Hiragino Mincho ProN W6.otf	Times.dfont
AppleGothic.ttf	Didot.dfont	Hoefler Text.dfont	TimesLTMM
AquaKanaBold.otf	Futura.dfont	Impact.ttf	Trebuchet MS Bold Italic.ttf
AquaKanaRegular.otf	Geeza Pro Bold.ttf	Kai.dfont	Trebuchet MS Bold.ttf
Arial Black.ttf	Geeza Pro.ttf	Keyboard.dfont	Trebuchet MS Italic.ttf
Arial Bold Italic.ttf	Geneva.dfont	LastResort.dfont	Trebuchet MS.ttf
Arial Bold.ttf	Georgia Bold Italic.ttf	LiHei Pro.ttf	Verdana Bold Italic.ttf
Arial Italic.ttf	Georgia Bold.ttf	LucidaGrande.dfont	Verdana Bold.ttf
Arial Narrow Bold Italic.ttf	Georgia Italic.ttf	MarkerFelt.dfont	Verdana Italic.ttf
Arial Narrow Bold.ttf	Georgia.ttf	Microsoft Sans Serif.ttf	Verdana.ttf
Arial Narrow Italic.ttf	GillSans.dfont	Monaco.dfont	Webdings.ttf
Arial Narrow.ttf	Hei.dfont	Optima.dfont	Wingdings 2.ttf
Arial Rounded Bold.ttf	HelveLTMM	Osaka.dfont	Wingdings 3.ttf
Arial Unicode.ttf	Helvetica LT MM	OsakaMono.dfont	Wingdings.ttf
Arial.ttf	Helvetica.dfont	Papyrus.dfont	ZapfDingbats.dfont
Baskerville.dfont	HelveticaNeue.dfont	Skia.dfont	Zapfino.dfont
BigCaslon.dfont	Herculanum.dfont	STHeiti Light.ttf	
Brush Script.ttf	Hiragino Kaku Gothic Pro W3.otf	STHeiti Regular.ttf	
Chalkboard.ttf	Hiragino Kaku Gothic Pro W6.otf	Symbol.dfont	

Note the multiple font file formats:
.ttf TrueType
.otf OpenType
.dfont Data fork TrueType suitcase
 (an older type of TrueType font used on Mac OS’ prior to OS 10.x)

How Are Fonts Managed?

Web-safe Fonts

Web-safe fonts are fonts likely to be present on a wide range of computer systems and are thus used by Web content producers to increase the likelihood that online content will be displayed as designed.

When visitors to a website do not have the specified font, their browser will attempt to select an alternative font from a “font-stack” which is a series of fallback fonts and generic font families specified in advance by the content producer. This is referred to as a “degradation” process.

The CSS code structure of a font-stack is shown below. The browser will attempt to render the appropriate text in the first listed font. In the event that the first font is not available, the browser will try to use the second font and so on.

```
p {
  font-family: Garamond, Palatino, Times, serif;
}
```

↑
When specifying a font stack in CSS, a fallback generic font family should be specified as a last resort. The possible generic font families are “serif”, “sans-serif”, “monospace”, “cursive”, and “fantasy”. The specific font used for each generic font family is stored in browser preferences. Most browsers allow the user to change the default generic font to any other font on their computer.

The fonts listed here can reliably be found on the three major computer operating systems. Most of the fonts in the grouping to the right ship with the operating systems themselves. The weighted median is adjusted to account for the number of people who use each operating system.

Some popular fonts, such as Helvetica, are almost always present on the Mac OS but rarely found on Windows or Linux. These fonts are unreliable choices for Web typography.

	Windows	Mac	Linux	Weighted Median
Courier	99.71 %	98.87 %	66.14 %	98.60 %
Verdana	99.76 %	97.46 %	62.66 %	98.50 %
Times	99.47 %	98.02 %	65.19 %	98.30 %
Arial	99.33 %	97.74 %	67.72 %	98.20 %
Trebuchet	99.38 %	94.63 %	62.03 %	98.00 %
Lucida	98.76 %	100.00 %	77.86 %	97.90 %
Georgia	99.04 %	95.76 %	62.66 %	97.80 %
Impact	99.00 %	91.24 %	61.08 %	97.50 %
Arial Black	98.52 %	94.07 %	62.66 %	97.20 %
Tahoma	99.90 %	79.10 %	0.00 %	97.00 %
Palatino	98.76 %	78.81 %	0.00 %	95.90 %
Arial Narrow	88.95 %	90.11 %	0.71 %	87.40 %
Century Gothic	88.04 %	40.40 %	0.00 %	83.90 %
...				
Helvetica	7.38 %	100.00 %	18.62 %	
Helvetica Neue	1.60 %	96.46 %	-	

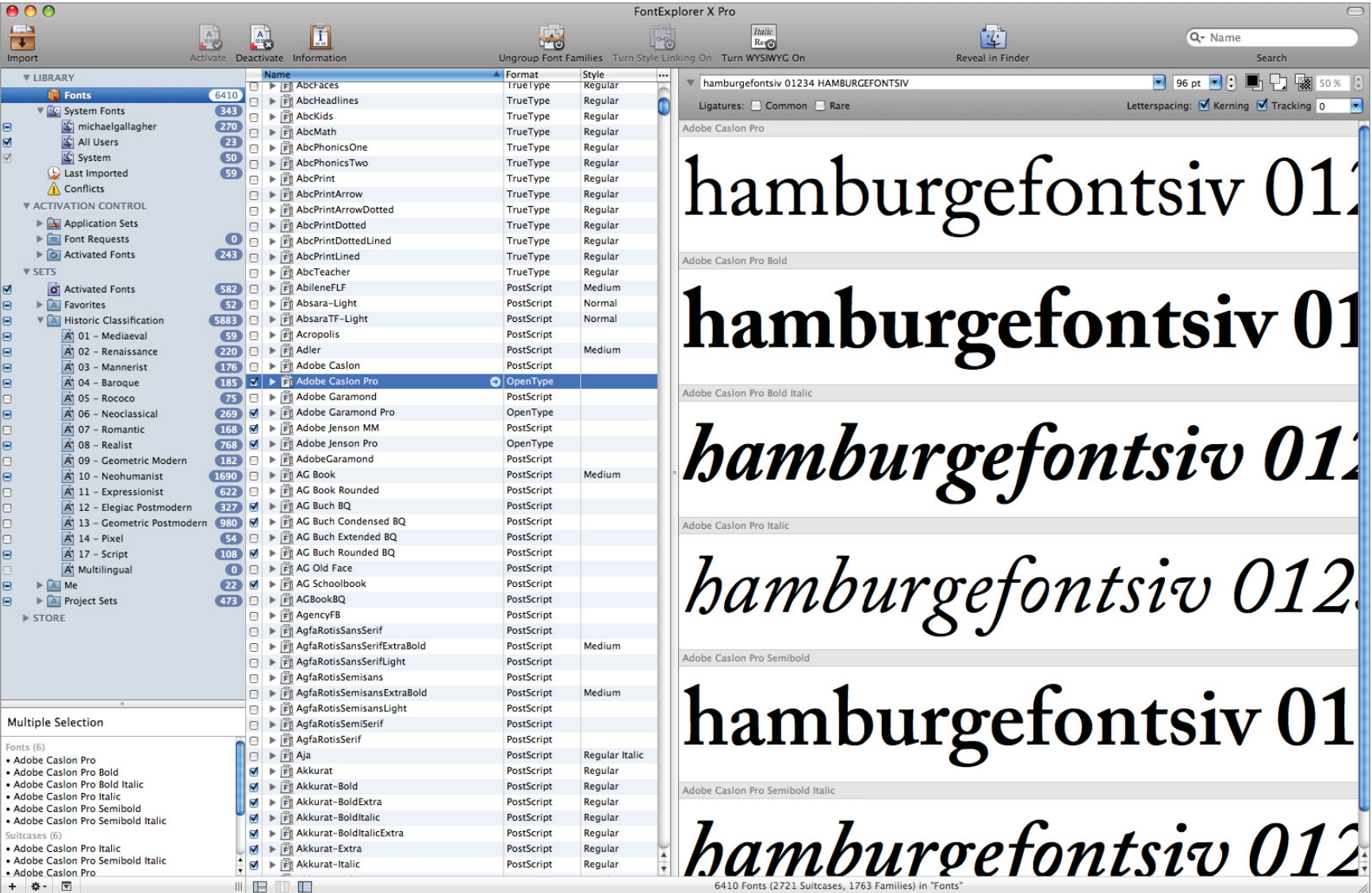
Sources:
www.mightymeta.co.uk/web-safe-font-cheat-sheet-v-2-including-google-font-api/
www.codestyle.org/css/font-family/sampler-CombinedResultsFull.shtml
support.apple.com/kb/ht1642 (list of fonts shipping with Mac OS 10.5)
www.microsoft.com/typography/fonts/product.aspx?pid=161 (list of fonts shipping with Windows 7)
www.apaddedcell.com/sites/apaddedcell.com/files/fonts-article/Linux.html (list of fonts that ship with Ubuntu Linux)

How Are Fonts Managed?

Font Management Software

Fonts are not some sort of inherent property of operating systems; they are digital files that are stored in a folder (or folders) and must be accessed by the operating system in order to generate glyph bitmaps. Simply having a font file on your computer's hard drive does not allow it to be accessed. The operating system must first index the font file, which is the most basic role of font management software. The operating system's font index is called the "font cache". A font cache allows the operating system to provide applications a complete list of all installed fonts and previews quickly in response to user's formatting choices.

Most operating systems' font management software (e.g. Font Book on the Mac OS) is very limited in capability. More advanced font management software allows users to sort and organize font files as well as turn fonts on or off, effectively adding and removing them from the operating system's font cache as needed. This can be especially important when a user's font library is very large – 5,000 fonts is not an exceptional number of fonts for professional designers to have on their computers. A large font cache can slow down a personal computer to a crawl.



A contemporary font management program (FontExplorer) used to organize a large font library.

How Are Fonts Managed?

Font Foundries

Companies that license or sell (and typically also design) fonts are called font foundries. Font foundries have existed as long as typefaces have been sold (prior to typefaces being sold, individual printers would design and cut their own type for private use). In the late 19th century there were a vast number of foundries of all sizes. However extensive consolidation took place in response to improved mechanical typesetting technology. This pattern would be repeated in the mid-twentieth century with the number of foundries increasing significantly only to see another wave of consolidation. Today, because of digital technology, there are thousands of micro-foundries, sometimes being run by just a single person.

For the most part, fonts are not sold, they are licensed. Fonts are typically licensed either individually (e.g. a user would license just Univers 55 Regular) or at a package discount for a group of fonts including some portion of a font family. Typically, a separate license must be acquired for every device that will use the font file; however some licenses allow for a single user to use the font on multiple machines provided they are the only one using it.

Most end user license agreements (EULAs) do not permit the user to re-sell, distribute, or modify the font file in any way. This distinction between selling and licensing fonts – along with what a licensee is permitted to do with the fonts – is extremely important in the realm of Web typography because most font EULAs explicitly prohibit font files from being posted to a Web server for remote access by webpages. Far fewer fonts have Web licenses available and these typically have to be purchased independently of a standard license.

Contemporary Font Foundries

- Bitstream
- Buro Destruct
- Commercial Type
- Dutch Type Library
- Elsner & Flake
- Feliciano
- Font Bank (Korea)
- Font Shop
- Fontomas
- Fountain
- Han Yang (Korea)
- Hoefler & Frere Jones
- House Industries
- Klim
- KLTF
- Kontour
- Letterror
- Lineto
- Linotype
- Monotype
- Morisawa (Japan)
- Neufville Digital Foundry
- Optimo
- OurType
- Porchez Typefonderie
- Process Type Foundry
- PSY/OPs
- Sandol (Korea)
- The Font Bureau
- The Foundry
- Thirstype
- Typotheque
- Underware
- Village
- Yoon (Korea)
- ...

Contemporary Type Designers and Fonts they have Designed

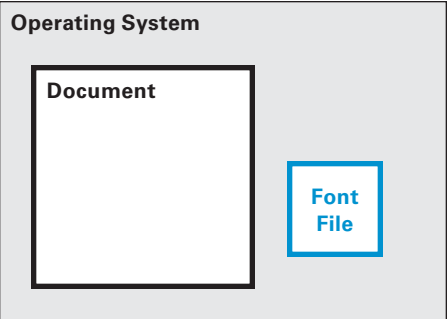
Designer	Fonts
Ahn Sang-Soo	I-Sang, Mano
Peter Bilak	Fedra, Eureka, Greta
Erik van Blokland	Federal, Trixie
Laurenz Br��nner	Akkurat, Circular
Matthew Carter	Verdana, Georgia, Galliard, Bell Centennial, Charter
Gavillet & Rust	Executive, Hermes
Tobias Frere-Jones	Gotham, Nobel, Interstate, Whitney, Retina
Jonathan Hoefler	Champion Gothic, Hoefler Text, Knockout, Verlag
Zuzana Licko	Mrs. Eaves, Filosofia, Matrix
Martin Majoor	Scala, Seria
Norm	Purple, Replica
Radim P��sko	Boymans, Fugue, Mitim, Mercury
Jean Fran��ois Porchez	Le Monde, Parisine, Costa
Fran��ois Rappo	Theinhardt, Didot Elder
Christian Schwartz	Amplitude, Bau, Los Feliz, Neutra
Robert Slimbach	Arno, Minion, Myriad
Fred Smeijers	Arnhem, Fresco, Sansa
Kris Sowersby	Founders Grotesque, National, Tiempos, Feijoa
Erik Spiekerman	Meta, Unit, Officina
Underware	Dolly, Fakir, Auto, Bello
Gerard Unger	Swift, Argo, Gulliver, Vesta

Font Embedding

Font embedding inserts a font file (or a subset of a font file) into a document, such as a Word document or PDF, to allow readers to see the document as formatted by the author. Font embedding is controversial because it is possible to unpack the document and extract the font file. Although this is not a trivial operation for most computer users, anyone with a type design program (e.g. Font Forge) can extract a font file from a PDF with relative ease.

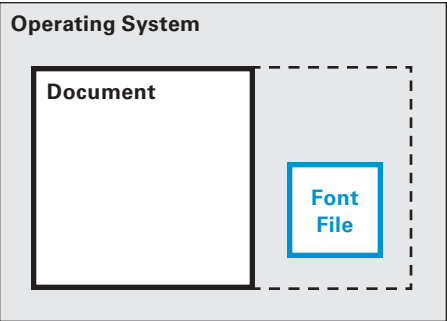
Font accessible to the operating system

Rendering the document as intended requires users to have a copy of the font on their computer. Without a copy of the font, the document will substitute the next specified font or what it determines to be an appropriate replacement if no alternative font has been specified.



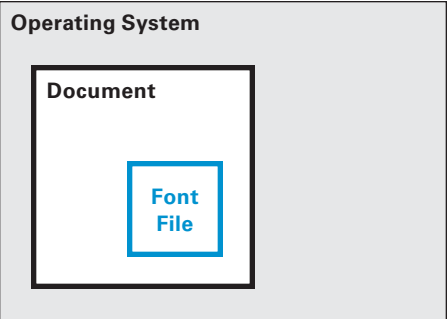
Font shipped with the document, but not *in* the document

The font file is shipped with the document; it is intended to help render. Many font licences specifically forbid this.



Font resident in the document

The font is embedded into the document. This method should increase the chances that the document renders as intended by the author. Often the font file is subsetting: embedding only the necessary characters to render the text in the document. Subsetting is a strategy to both reduce file size and prevent piracy.



Font Substitution

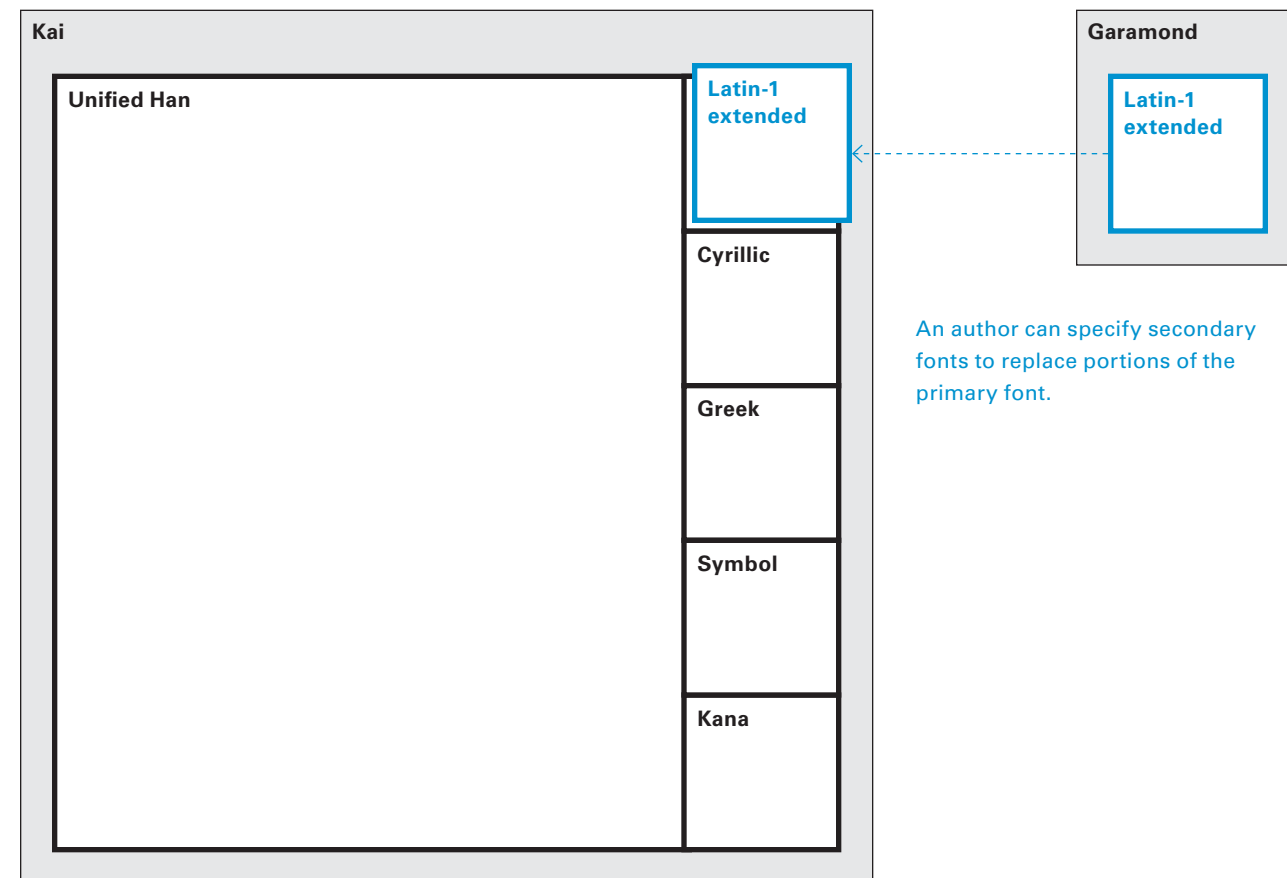
Font substitution is the process of using one font in place of another when the intended font either is not available or does not contain glyphs for the required characters. Font substitution can occur on a glyph-by-glyph basis. Not all systems perform font substitution, and not all systems that perform font substitution are able to substitute for missing characters; some are only capable of substituting for missing fonts. Most major Web browser can perform font substitution with the exception of versions of Internet Explorer older than version 7.



Font substitution can be controlled somewhat by Web designers who can specify a preferred series of fonts, to use when displaying a page.

Font Linking

Font linking is a technique for associating two or more fonts, usually as a way to select what font is used for particular languages. For example, most Chinese fonts include Latin characters needed to display English and other western writing systems. However most of these characters are not very well drawn. Font linking helps guarantee that each unique writing system in a block of text displays and prints at the highest possible level of quality.



How Is Text Formatted?

How Is Text Formatted?

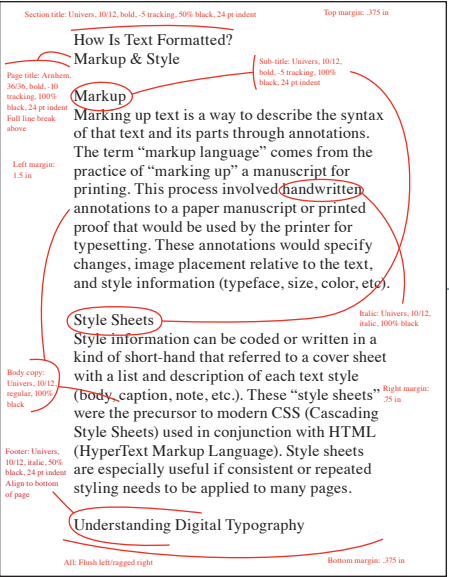
Markup & Style

Before computers, authors created manuscripts either by hand or typewriter. The graphic designer responsible for the look of the final printed piece would specify the font, size, leading, tracking, alignment, and more by writing specifications, known as “markup”, directly on the manuscript. Skilled typesetters would follow those specifications to either set metal type or direct their formatting while operating a phototypesetter or digital typesetter.

Specifying every detail of every style change in a book manuscript or other long document is repetitive and tedious, and typically designers have in mind just a few styles anyway. Thus a more efficient short-hand naturally evolves – a table of styles and abbreviations or codes – style sheets. Codes can include names, letters, numbers, and even colors – using colored highlights makes markup very efficient.

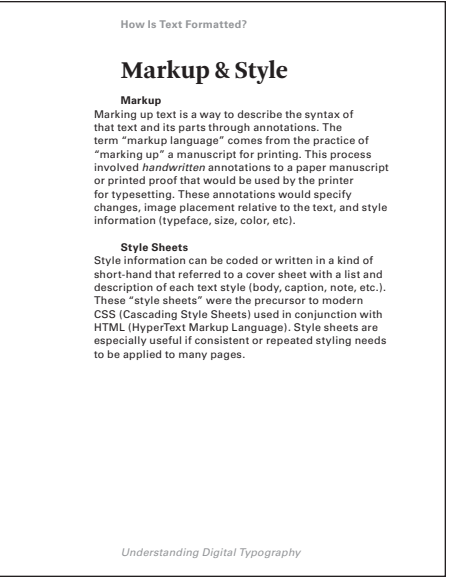
The invention of the style sheet allowed the structure and organization of the manuscript to be separated from specific style choices. That means styles could be altered throughout the manuscript simply by changing the style sheet.

Marked-up Manuscript with Style Notation

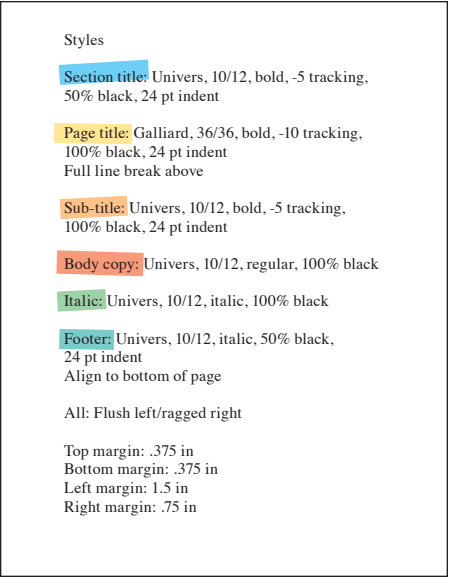


Here, detailed specifications are explicitly embedded in the manuscript.

Styled Document

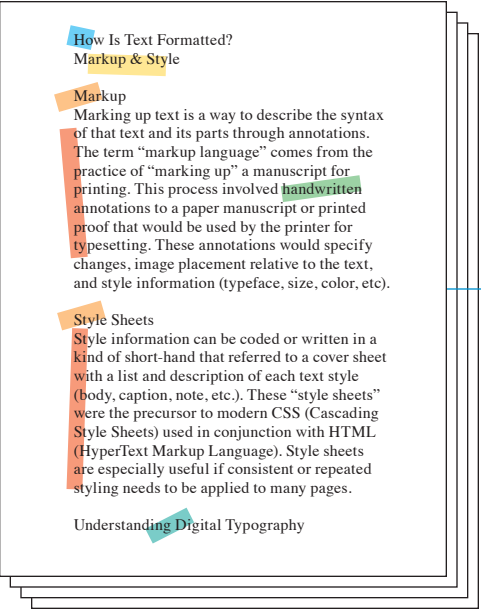


Style Sheet



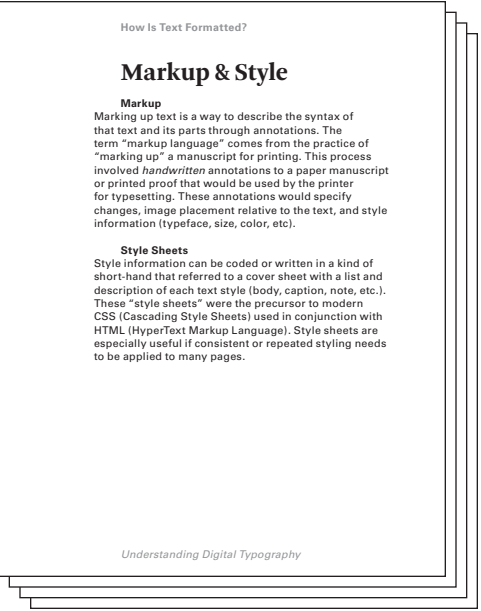
Here the style is coded by color. (Letters or numbered could also be used.) Consolidating all the detailed specifications in this way makes global changes easy.

Marked-up Manuscript



Here only a code indicating the style is embedded, the detailed specification is in the style sheet.

Styled Document



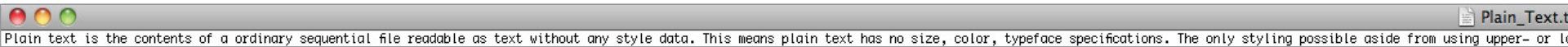
How Is Text Formatted?

Plain Text

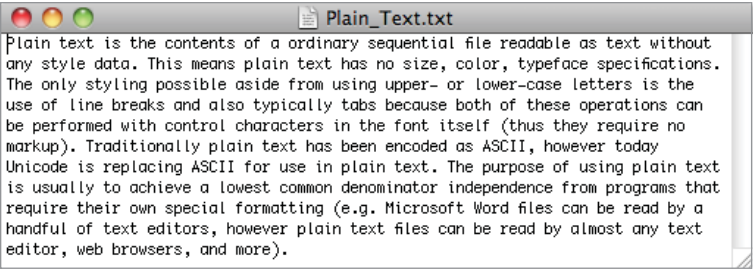
Plain text is the content of an ordinary sequential text file without any style data beyond the choice of character (upper or lower case), and use of tabs and line breaks, both of which are control characters found in the font itself. Plain text has no size, color, or other typeface specifications – any of these attributes you may see applied to plain text are provided by the application to make the text legible.

Plain text is highly portable and, unlike application specific formats such as .indd and .doc, can be opened and read with a wide variety of applications. Plain text is the digital analog of the unformatted, hand- or typewritten manuscripts of the pre-computer era. Like those manuscripts, plain text can be marked up with a series of tags to change its appearance and layout. But unlike text in print, digital markup may not specify typographic settings in detail – or at all. The final choice of font, size, style, etc. may be uniquely defined by the viewer application or even by the user.

This raises an important issue – **who should control how content, especially text, is displayed?** In print, the graphic designer’s decisions are fixed. Onscreen, designers have limited control over the display capabilities of the devices readers use to view the text. This ambiguity may be acceptable for certain types of content such as online periodicals. Content that requires more careful and controlled formatting may be restricted to dedicated, platform-specific apps. **The appropriate balance of designer control and user flexibility remain a subject of debate for webpages and electronic books.**

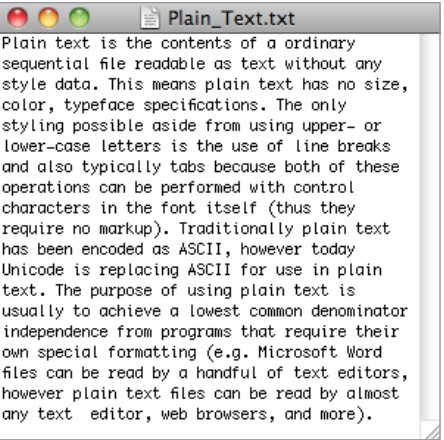


Plain text is the contents of an ordinary sequential file readable as text without any style data. This means plain text has no size, color, typeface specifications. The only styling possible aside from using upper- or lower-case letters is the use of line breaks and also typically tabs because both of these operations can be performed with control characters in the font itself (thus they require no markup). Traditionally plain text has been encoded as ASCII, however today Unicode is replacing ASCII for use in plain text. The purpose of using plain text is usually to achieve a lowest common denominator independence from programs that require their own special formatting (e.g. Microsoft Word files can be read by a handful of text editors, however plain text files can be read by almost any text editor, web browsers, and more).

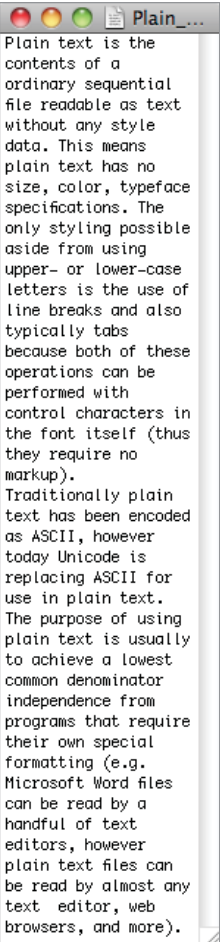


Plain text is the contents of an ordinary sequential file readable as text without any style data. This means plain text has no size, color, typeface specifications. The only styling possible aside from using upper- or lower-case letters is the use of line breaks and also typically tabs because both of these operations can be performed with control characters in the font itself (thus they require no markup). Traditionally plain text has been encoded as ASCII, however today Unicode is replacing ASCII for use in plain text. The purpose of using plain text is usually to achieve a lowest common denominator independence from programs that require their own special formatting (e.g. Microsoft Word files can be read by a handful of text editors, however plain text files can be read by almost any text editor, web browsers, and more).

A plain text file in Mac Text Edit. The font and size are not specified by the file itself, rather those attributes are provided by the application. The column width is determined by the width of the application window; the text can reflow to be any width desired because this information is not encoded into the text.



Plain text is the contents of an ordinary sequential file readable as text without any style data. This means plain text has no size, color, typeface specifications. The only styling possible aside from using upper- or lower-case letters is the use of line breaks and also typically tabs because both of these operations can be performed with control characters in the font itself (thus they require no markup). Traditionally plain text has been encoded as ASCII, however today Unicode is replacing ASCII for use in plain text. The purpose of using plain text is usually to achieve a lowest common denominator independence from programs that require their own special formatting (e.g. Microsoft Word files can be read by a handful of text editors, however plain text files can be read by almost any text editor, web browsers, and more).



Plain text is the contents of an ordinary sequential file readable as text without any style data. This means plain text has no size, color, typeface specifications. The only styling possible aside from using upper- or lower-case letters is the use of line breaks and also typically tabs because both of these operations can be performed with control characters in the font itself (thus they require no markup). Traditionally plain text has been encoded as ASCII, however today Unicode is replacing ASCII for use in plain text. The purpose of using plain text is usually to achieve a lowest common denominator independence from programs that require their own special formatting (e.g. Microsoft Word files can be read by a handful of text editors, however plain text files can be read by almost any text editor, web browsers, and more).

How Is Text Formatted?

Evolution of the Typesetting “Stack”

	1456	1884	c.1954	c.1980	c.1985	c.1995	c.2000
	Compositor	Machine Typesetter	Photo Typesetter	Digital Typesetter	Digital Imagesetter	Direct-to-Plate	High Speed Inkjet
Source	Handwritten manuscript. (The compositor may also be the designer.)	Typed manuscript marked up by a designer.	Typed manuscript marked up by a designer.	Text delivered on disk accompanied by a printed manuscript marked up by a designer.	Electronic file delivered on disk, already formatted by a designer.	Electronic file delivered via email, already formatted by a designer.	Electronic templates delivered via email, already formatted by a designer.
Composition Method	A person selects individual pre-cast metal letters and arranges them by hand into lines of text.	Typing letters on the keyboard of a typesetting machine causes a series of moulds to be assembled and filled with molten lead, casting individual letters or complete lines of text.	Early photo typesetters replace moulds with negatives. Later devices arrange negatives on glass plates or film strips. Light shown through the negatives exposes high contrast photo paper, creating typeset proofs. Early devices are driven by paper tape; later devices are driven by computer.	Once photo typesetters are driven by computers, the next step is replacing the film negative with a digital image. Digital typesetters use CRTs or lasers to create images. In most cases they expose photo paper just as photo typesetters do. (Some digital typesetters employ xerography, but the quality is low.)	Digital imagesetters are essentially digital typesetters without composition systems. Composition once done by people skilled at typesetting is now done by designers, who may know less of the craft. Image setters can expose paper or film, creating the potential to skip paste up and go “direct to film”.	Plate imaging devices are like imagesetters except that they expose plates instead of photo paper or film. That means no negatives are required.	Composition takes on a very different form. The process assumes each “print” will be customized. Designers must decide which elements will appear on all copies and which elements will vary from copy to copy. Lists of recipients must also be prepared.
Page Layout Method	Lines of text are tightly bound into blocks.	Lines of text are tightly bound into blocks.	Proofs are cut up, arranged, and pasted up on art boards called “mechanicals”.	Proofs are cut up, arranged, and pasted up on art boards called “mechanicals”.	Designers layout whole documents before sending them to the imagesetter.	Designers layout whole documents before sending them to the printing company.	Designers create templates, assemble variable elements, and define rules for inserting them.
Form Creation Method	A block or blocks are locked directly into the bed of a press or locked into a moveable frame which is locked into the bed of a press.	A block or blocks are locked directly into the bed of a press or locked into a moveable frame which is locked into the bed of a press.	Mechanicals are photographed in large process cameras to create high contrast negatives. Negatives are taped or “stripped” into paper or plastic forms. The forms are sandwiched against a raw printing plate which is exposed with an arc lamp and then developed.	Mechanicals are photographed in large process cameras to create high contrast negatives. Negatives are taped or “stripped” into paper or plastic forms. The forms are sandwiched against a raw printing plate which is exposed with an arc lamp and then developed.	No mechanicals are needed. Negatives are taped or “stripped” into paper or plastic forms. The forms are sandwiched against a raw printing plate which is exposed with an arc lamp and then developed.	Using plate preparation software, experts at the printing company arrange pages from the designer’s formatted document so that they are in the right position for printing.	Some additional formatting may be required by experts who operate the press.
Reproduction Method	Copies are printed directly from the hand set type.	Copies are printed directly from the machine set type. In the late stages of “hot type”, only 2 or 3 proofs were printed. Pages were then laid out and prepared for printing as described in the next stack.	Copies are printed from the printing plate.	Copies are printed from the printing plate.	Copies are printed from the printing plate.	Copies are printed from the printing plate.	A computer controls a complex process that takes information from a list of recipients, matches it with rules, and inserts variable elements into templates to compose a page just in time for printing. The computer then rasterizes the page and sends the raster file to the printer.
Method Name	Letterpress.	Letterpress. Offset Lithography.	Offset Lithography.	Offset Lithography.	Offset Lithography.	Offset Lithography.	Inkjet.
		Gray indicates a step unchanged from previous stack.					

How Is Text Formatted?

TeX

TeX (pronounced /tek/) is a typesetting system designed by Donald Knuth beginning in 1977 to typeset his own manuscript after being displeased by the proofs he received from a publisher. Today TeX is used primarily for typesetting papers that involve mathematical formulas – it is popular in academia, especially mathematics, engineering, physics, computer science, economics, statistics, and quantitative psychology. Text markup in TeX uses commands (usually specified by a backslash) that are applied to one or more pieces of text (indicated by curly brackets). The base TeX system understands about 300 commands, called primitives. However, these low-level commands are rarely used directly by users, and most functionality is provided by format files (pre-built collections of macro commands and formatting data). The most widely used format file is called LaTeX – it incorporates document styles for books, letters, slides, etc., and adds support for referencing and automatic numbering of sections and equations.

While TeX can be written in any text editor, it requires a special editing program to compile and output files based on the code. The TeX system was designed to work with Metafont, the font file format Knuth also designed, but TeX can also use PostScript fonts.

Simple Text Formatting in TeX

Code:

The dog is classified as \textit{Canis lupus familiaris} in taxonomy.
\bye

Output:

The dog is classified as *Canis lupus familiaris* in taxonomy.

All markup languages employ a code character to differentiate instructions from primary text. These characters are called “command” characters, “special” characters, or “delimiters”. Without them, text markup would not exist in its current form.

The code \textit is used in TeX to indicate that any text that follows (and surrounded by curly brackets) will be italic.

Mathematical Equation Formatting in TeX

Code:

The quadratic formula is \$-b \pm \sqrt{b^2 - 4ac} \over 2a\$
\bye

Output:

The quadratic formula is
$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In the TeX system, mathematics mode is entered by using a \$ character, after which the user can enter a formula in TeX semantics and then close with another \$.

How Is Text Formatted?

PDF

The Portable Document Format (PDF), created in 1993 by Adobe, is used for representing two-dimensional documents in a manner independent of application software, hardware, or operating system.

PDFs are built on three technologies: a subset of the PostScript page description language for generating layout and 2D graphics, a font-embedding/replacement system to allow fonts to travel with the document, and a storage system to bundle elements into a single file (with data compression where appropriate). The version of PostScript used in PDF is simplified to remove *if* and *loop* commands while graphics commands such as *lineto* remain. However PDF supports functions such as transparency, compression, and password protection that PostScript does not.

PDF encoding of text. Each line of text in a column is described as a distinct section.

The numbers describe the width of each character.

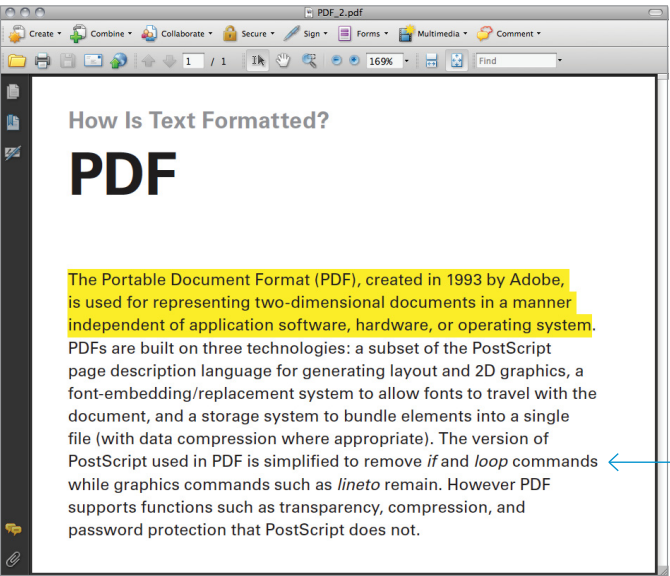
Changes in text formatting, such as italicization, create a new text chunk.

PDF Formatting

Code:

```
(PostScript used in PDF is simplified to remove )
[6.00792 6.17693 5.237 3.46892 6.78493 5.065 3.92892 2.73799 6.22693 3.27992 2.73 6.16693
5.10899 5.65401 6.05992 2.73 2.71501 6.05992 2.73 6.14592 7.37685 5.51001 2.73 2.81099
4.95 2.73 5.03502 2.724 9.41785 6.14893 2.7 2.81699 3.07147 3.07147 5.65399 6.05991
2.73001 3.2919 6.05994 2.73001 3.94791 5.608 9.45386 6.02292 5.48801 5.51001 0 ]xsh
/HLTFWB+Univers-Oblique*1
[101{/.notdef}rp /e /f 2{/.notdef}rp /i 2{/.notdef}rp /l /.notdef
/n /o /p 3{/.notdef}rp /t 139{/.notdef}rp]
HLTFWB+Univers-Oblique nf
HLTFWB+Univers-Oblique*1 [10 0 0 -10 0 0 ]msf
329.395 290 mo
(if)
[2.72 0 ]xsh
HLTFWA+Univers*1 [10 0 0 -10 0 0 ]msf
335.395 290 mo
( and )
[2.72998 5.55099 6.13394 6.05994 0 ]xsh
HLTFWB+Univers-Oblique*1 [10 0 0 -10 0 0 ]msf
358.6 290 mo
(loop)
[2.75998 6.13193 6.13794 0 ]xsh
HLTFWA+Univers*1 [10 0 0 -10 0 0 ]msf
379.69 290 mo
( commands )
[2.73001 5.11597 6.12396 9.41483 9.43484 5.55103 6.13391 6.22394 4.99997 0 ]xsh
```

Output:



How Is Text Formatted?

HTML

The backbone of the Web is **HyperText Markup Language (HTML)**. HTML was invented in 1990 by Tim Berners-Lee based on earlier research he had begun in 1980 while working at CERN. The work Berners-Lee developed in 1980 was done so that CERN scientists would have a way to display physics papers on screen. However Berners-Lee was neither a computer scientist nor a typographer, and perhaps because of this HTML has a number of strange features – content and presentation are conflated, all markup has to be repeated for all elements, etc.

As the name implies, **HTML is a way to mark-up content and indicate to a computer what it is made of, e.g. paragraphs, headings, images, links** (this is often referred to as “semantic HTML”). HTML is composed of elements that are defined by tags that typically take the form of an opening tag and a closing tag, inside of which there is content (there are some tags that are self-closing). Web browsers read HTML pages and compose them into visual and audible webpages. The browser does not display HTML tags, but uses them to interpret the content of the page.

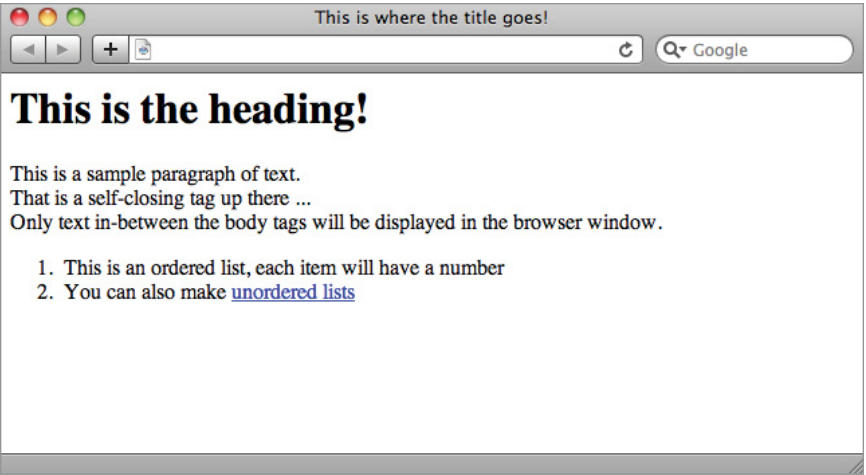
The HTML only describes what the content is – it does not dictate how it should look. In this example, browser defaults were referenced to for defaults fonts, header sizes, list indents, etc.

HTML Formatting

Code:

```
<html>
  <head>
    <title>This is where the title goes!</title>
  </head>
  <body>
    <h1>
      This is the heading!
    </h1>
    <p>
      This is a sample paragraph of text. <br />
      That is a self-closing tag up there ... <br />
      Only text in-between the body tags will be displayed in the browser window.
    </p>
    <ol>
      <li>This is an ordered list, each item will have a number </li>
      <li>You can also make <a href="http://www.unorderedlists.com/">unordered lists</a></li>
    </ol>
  </body>
</html>
```

Output:



The most remarkable thing about HTML is linking – embedding the address or path to another page. For linking to work, the other page must be resident on the user’s computer or the computer must be connected to a network.

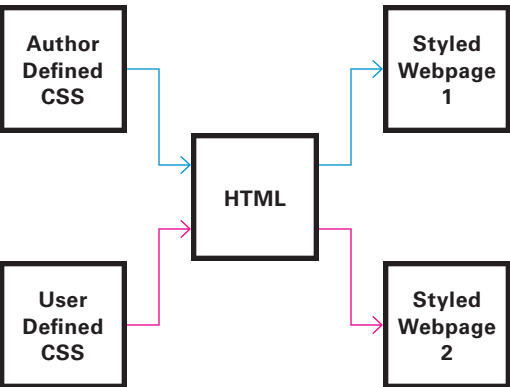
How Is Text Formatted?

CSS

HTML is used to categorize the contents of a webpage; Cascading Style Sheets (CSS) define the look and formatting of the content. CSS was designed to enable the separation of document from document presentation. In HTML, an <h1> tag simply tells the browser “this text is a heading”, it says nothing about what that heading should look like. CSS can be used to define the font, size, color, position, etc. of any type of tag.

CSS has a simple syntax that allows the writer to assign style data to standard HTML tags (<h1>, <p>, etc.) or create their own “selectors” that can be applied to HTML tags. These selectors can have a wide variety of properties, each with a defined value. It should be pointed out that CSS can be used to style languages other than HTML such as XHTML, SVG, XUL, and more.

As the name implies, CSS *cascades*. Style data is prioritized (or “weighted”) to determine what style rule applies to an element if more than one rule matches against a particular element – more specific rules are given more weight, so if two selectors apply to the same element, the one with higher specificity wins. This also allows multiple style sheets to be used for the same HTML document. Allowing for multiple style sheets is especially important because it provides a way for users to override an HTML documents given style with their own, customized style sheet.



CSS Formatting

Code:

The CSS code structure.

```
selector {
    property: value;
}
```

Sample CSS code for the <h1> HTML element.

```
h1 {
    margin: 15px 30px 15px 30px;
    font-family: Georgia, Times, serif;
    font-size: 30px;
    font-style: italic;
    font-weight: normal;
    letter-spacing: 1px;
    color: #090;
}
```

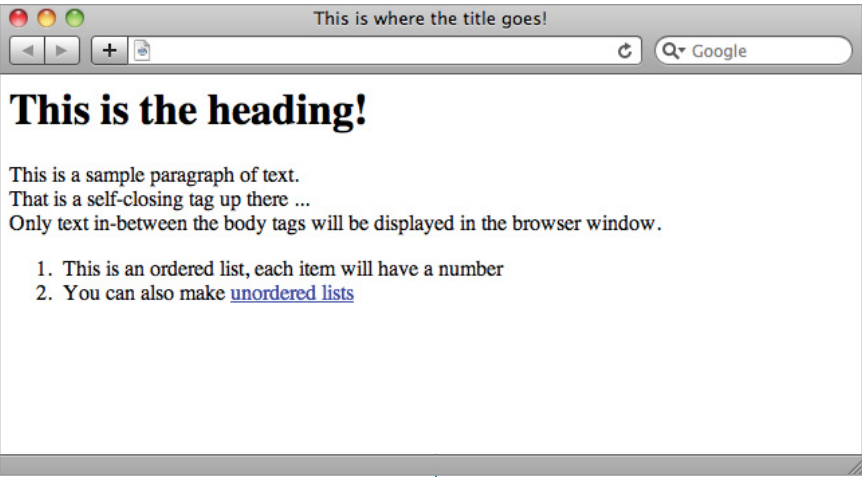
Sample CSS code for a writer defined selector.

```
.newStyle {
    margin: 15px 30px 18px 30px;
    padding: 16px 0 0 0;
    border-top: 1px solid #000;
    font-family: Georgia, Times, serif;
    font-size: 16px;
    line-height: 24px;
}
```

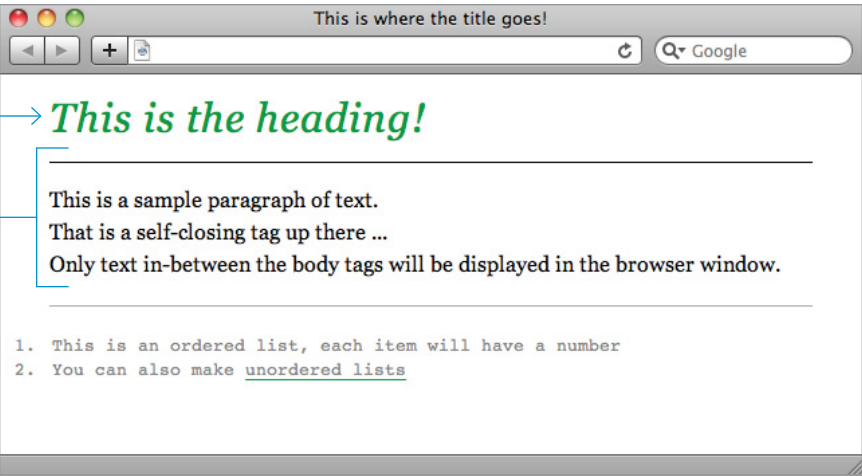
This style can be applied to HTML elements by adding a modifier to the tag.

```
<p class="newStyle">
```

Output:



No styling
(style data defaults supplied
by the application)



Styling applied

The screenshots use the same HTML code from page 87.

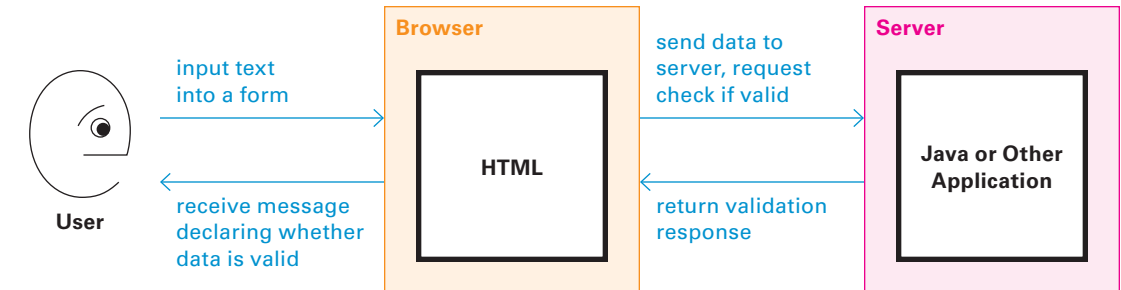
How Is Text Formatted?

JavaScript

JavaScript is a scripting language (a lightweight programming language) **designed to add greater interactivity to HTML pages**. JavaScript cannot be used on its own but must work in tandem with HTML. Its primary advantage is that it **works on the client side** (i.e. in the browser, rather than on the server), and this allows for the authoring of **dynamic behaviors that can be triggered by user actions** (e.g. changing the background color of a document depending on where the mouse is located on screen or validating user input text before it is sent to the server).

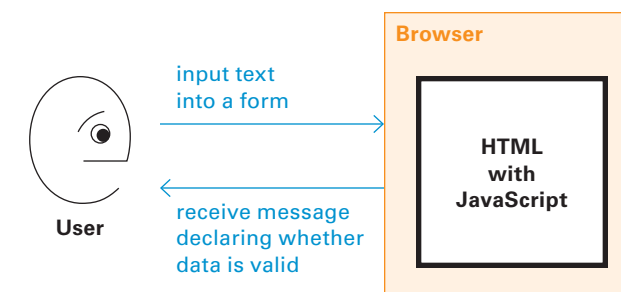
HTML Only

In order to validate text entered into a form, the browser has to send the data to a server, which checks the data and then returns a response. Further text entry requires the procedure to be repeated. Validation requests are sent when the user presses “enter”. This process can be slow and frustrating.



HTML with JavaScript

With JavaScript, text can be validated continuously in the browser on a character-by-character basis without any server queries. Using JavaScript in this way reduces lag and makes the page or application more responsive.

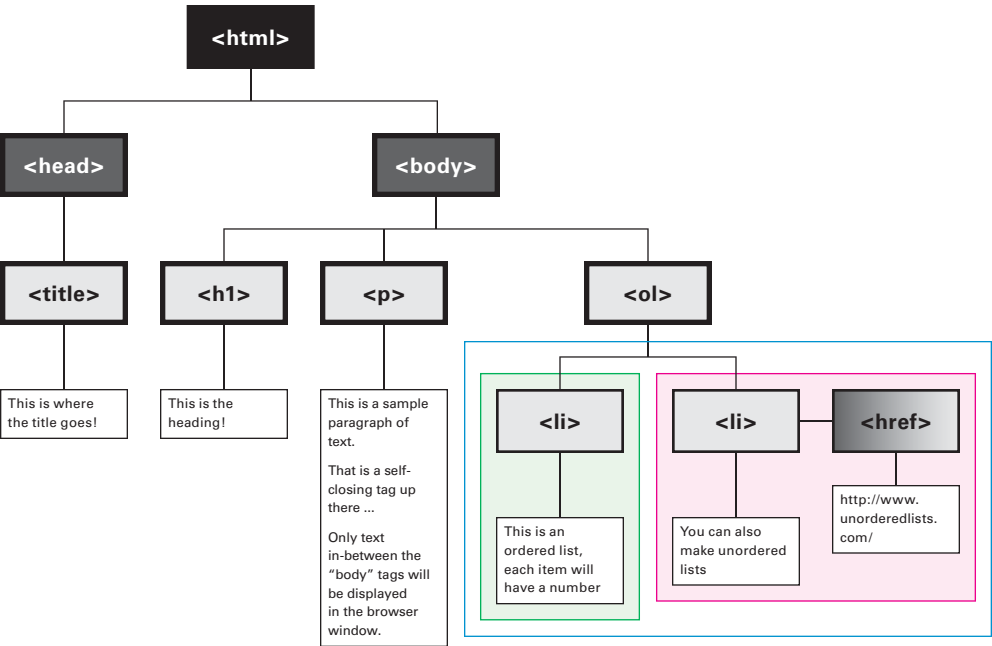


How Is Text Formatted?

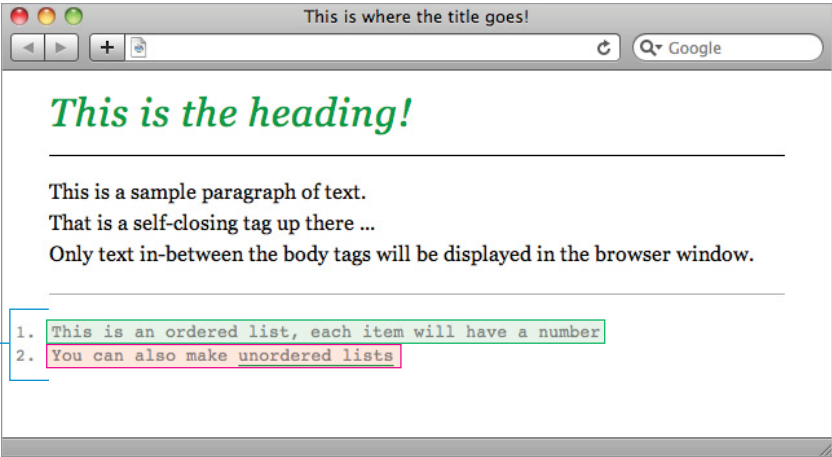
DOM

Dynamic behavior on a webpage means that something on a given webpage changes. Modern webpages use scripts written in JavaScript (see previous page) to alter the HTML or CSS. However, for the script to work, it must have a way of addressing the HTML or CSS – it must know what the elements of a page are called and where to find them. **The Document Object Model (DOM) is a standardized format that allows programs and scripts to dynamically access and update the content, structure, and style of documents.** The DOM views documents as a tree-structure, and all HTML documents received by a Web browser are parsed into a DOM tree before they can be rendered on screen. The tree is composed of element nodes that begin with the document type (<html>), go through the parts of the document (<h1>, <p>) and end in the content itself (the words between <p> tags). The DOM defines objects and properties for all elements in the document and methods to access them.

DOM Tree and its Corresponding Webpage



This example DOM tree uses the HTML code from the page 87.

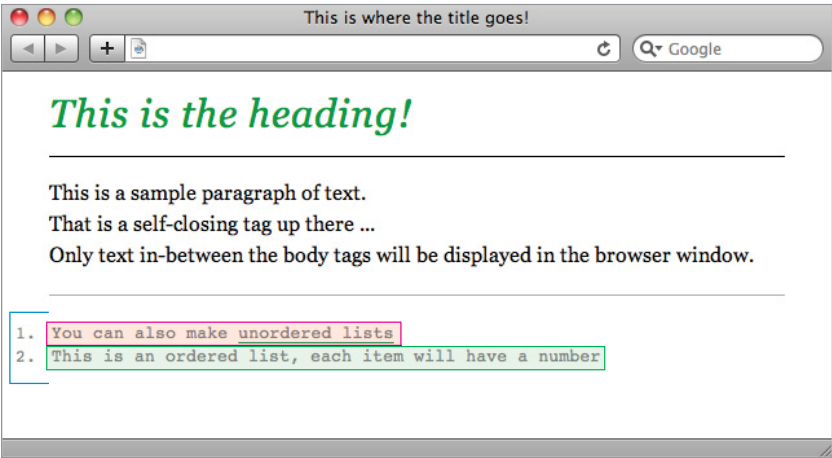
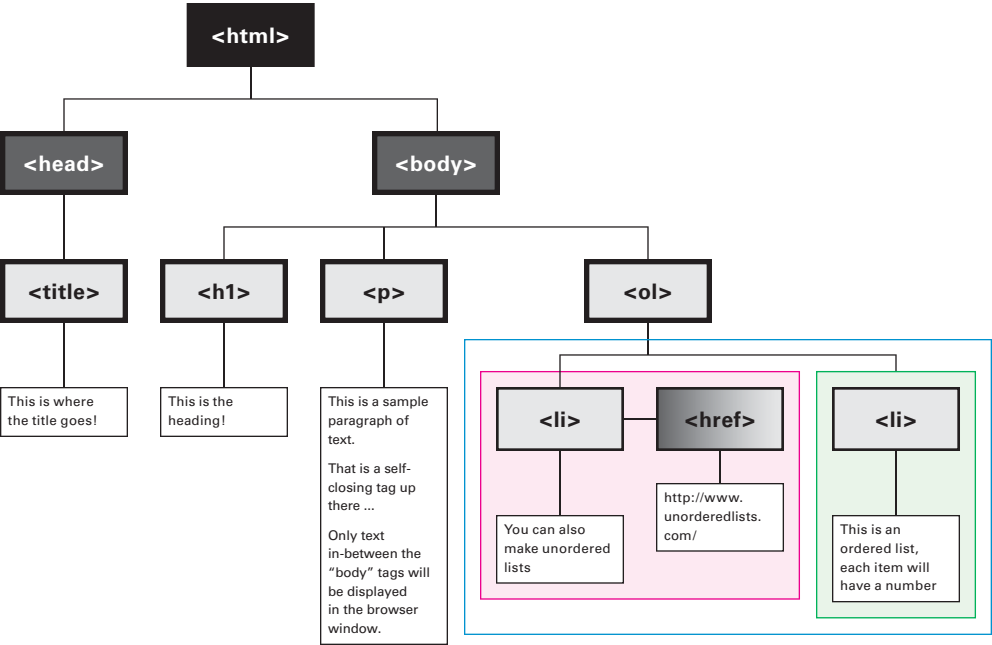


The DOM allows for dynamic behavior in the browser. For example, a JavaScript function could be written to dynamically re-order elements in a list. In this example, the following pseudo-code could be activated:

```
$(list).sortBy("reverse");
```

to allow a user to reverse the order of the list elements.

Modified DOM Tree and its Corresponding Webpage

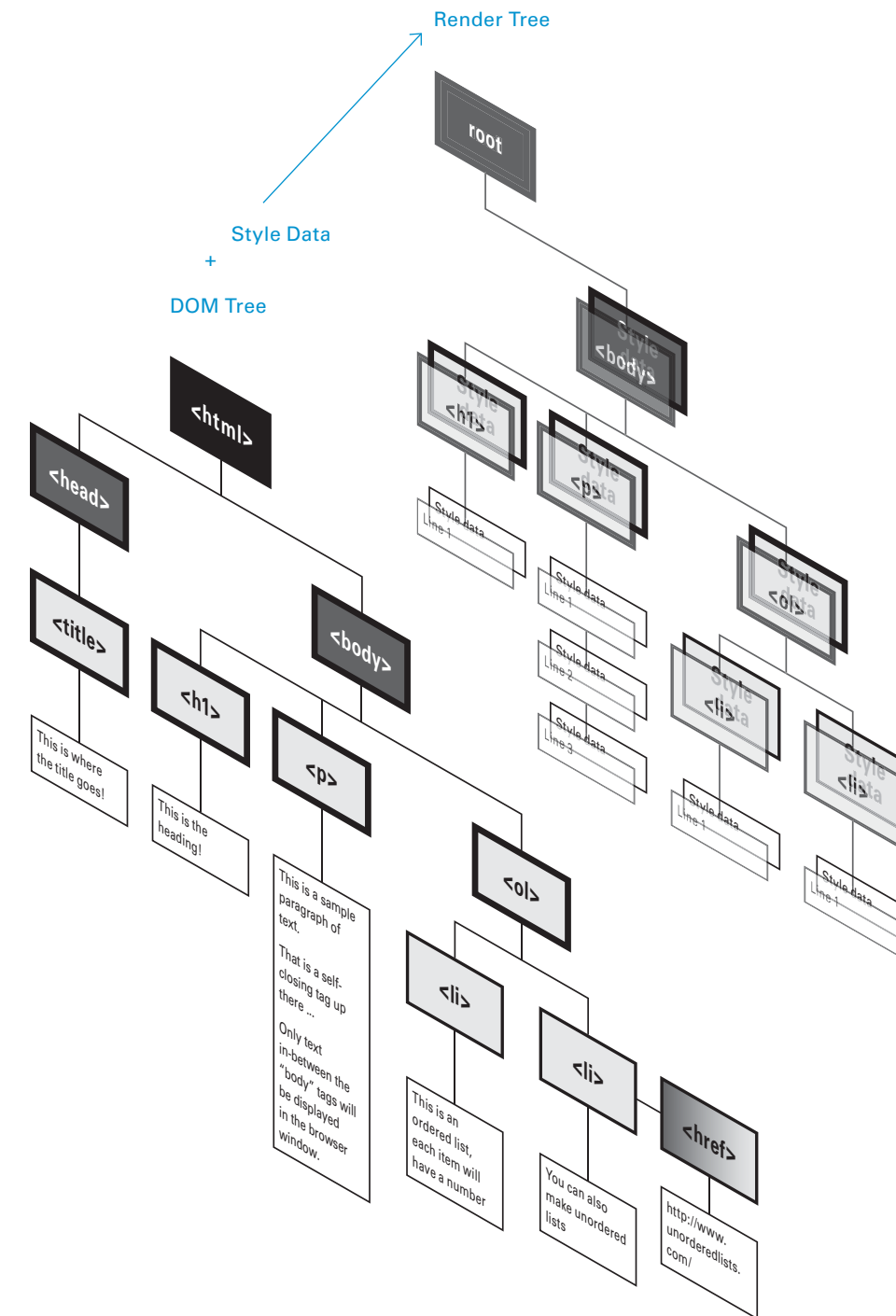


How Is Text Formatted?

Render Tree

The render tree is built from the DOM tree. The layout engine creates a parallel tree to the DOM by attaching style data to DOM elements to create render objects. The render tree is **used to instruct the rendering engine how to paint the contents of the DOM tree**. The render tree also instructs the rendering engine on what order to render elements.

Not all elements of the DOM tree are part of the render tree. Invisible elements, such as the `<head>` element are not painted on screen, and therefore do not need to be part of the render tree. The render tree also does not contain element attributes, such as link destinations.

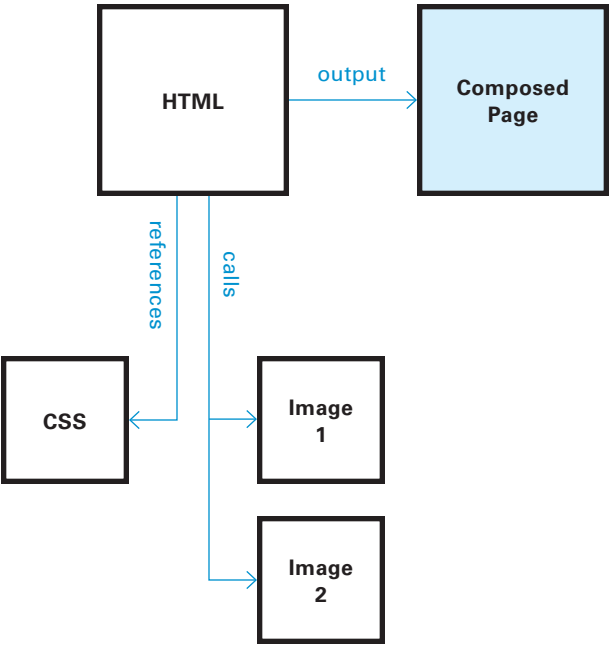


How Is Text Formatted?

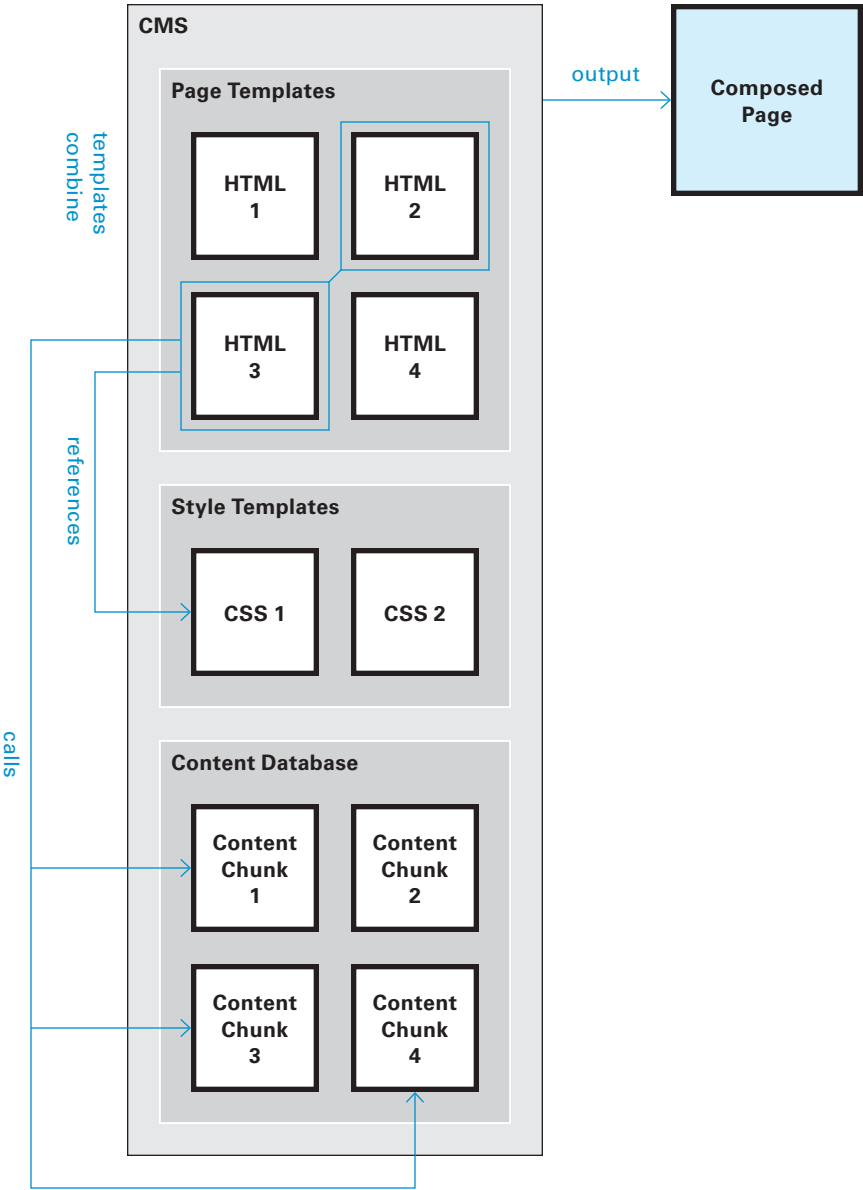
CMS

A content management system (CMS) is a method of managing content, workflow, and collaboration. Content management systems range from blogging software to corporate intranets. The core notion of a CMS is that it simplifies the production and publication of content by centralizing resources. For instance, when a website has three pages of HTML with some linked images, it is relatively easy to update the page layout or content chunks one-by-one. On the other hand, if a website has three hundred pages, updating the layout or a repeated piece of content for each page one-by-one would be a prohibitively time consuming task. Content management systems allow an author to automate this process – page layout templates provide a tool for making global layout changes and content templates provide a tool for standardizing content structure. A secondary benefit of content management systems is that they allow authors less versed in markup and programming languages to manage and publish content without having to learn a large amount of technical knowledge.

HTML with Linked Images and Style Resources



Content Management System



How Is Text Formatted?

XML

A markup language is way of writing and annotating information. Extensible Markup Language (XML) is a meta-language consisting of a few simple rules you can combine to define your own markup languages.

Tags are the atomic unit of XML and each has a name, attributes that describe the tag, and content. The content of a tag can be either data or more tags – nested tags allow for the formation of hierarchal structures.

XML-based languages (or, “schemas”) specify what tag names are allowed, which attributes are allowed on each tag, and how the document can be structured. XML schemas must be defined for an application to parse them correctly, and this is the role of the “schema document”. An XML schema document can come in several forms, most notably DTD (Document Type Definition) and W3C XML Schema – both serve the same purpose but W3C XML Schema is considered more powerful because it can be more specific than DTD, is namespace aware, and is itself written in XML.

XML was originally derived from HTML, and though the current HTML specification (HTML5) is heavily influenced by XML, it is not technically XML-based. There are many XML-based markup languages used commonly today. Some examples are:

- RSS (Really Simple Syndication), a format for marking up feeds of syndicated content. Where HTML may be used to mark up an article on a website, RSS would be used to mark up a preview of every article published by the site.
- SOAP (Simple Object Access Protocol), a protocol format for exchanging information between a Web server and a Web application.
- DITA (Darwin Information Typing Architecture), a modular format based on discrete modular “topics”, primarily used for reference documents.

Most office productivity tools (e.g. Microsoft Word and Excel, Apple’s iWork, and Open Office) have adopted XML-derived file formats. An advantage of defining formats within the XML standard is that it allows for greater inter-application portability.

Basic XML Rules

<tag attribute=“value”>Content</tag>



The opening tag can have attributes – it is possible to have more than one attribute. Content is nested between the opening and closing tags.

XML Implementation: RSS

```
<?xml version=“1.0” encoding=“UTF-8” ?>
<rss version=“2.0”>
<channel>
  <title>RSS Post Title</title>
  <description>This is an example of an RSS feed</description>
  <link>http://www.someexamplessdomain.com/main.html</link>
  <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
  <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
  <item>
    <title>Example entry</title>
    <description>Here is an interesting description.</description>
    <link>http://www.wikipedia.org/</link>
    <guid>unique string per item</guid>
    <pubDate>Mon, 06 Sep 2009 16:45:00 +0000 </pubDate>
  </item>
</channel>
</rss>
```

How Is Text Formatted?

Dublin Core

Dublin Core is a metadata specification for providing cataloguing information for physical objects such as books, digital materials such as texts, videos, sounds, or images, and composite media such as webpages. The specification has two levels: simple and qualified. Dublin Core is the standard metadata specification in the fields of library science and computer science.

Simple Dublin Core

There are 15 metadata elements in the simple set.

1	Title	Proper name of the resource
2	Creator	Author, artist, photographer, or illustrator
3	Subject	Keywords or phrases
4	Description	A textual description or abstract
5	Publisher	Publishing house, a university department, or a corporate entity
6	Contributor	For example: editor, transcriber, and illustrator
7	Date	A date associated with the creation or availability
8	Type	The category of the resources, such as home page, novel, poem, etc.
9	Format	The data format and dimensions (e.g. size, duration) of the resource
10	Identifier	A number used to uniquely identify the resource, such as URL and ISBN
11	Source	Information about a second resource from which the present resource is derived
12	Language	The language of the intellectual content of the resource
13	Relation	An identifier of a second resource and its relationship to the present resource
14	Coverage	Spatial coverage refers to a physical region which the resource is about
15	Rights	A rights management statement

Qualified Dublin Core

There are 3 additional metadata elements in the qualified set.

- 1 Audience
- 2 Provenance
- 3 RightsHolder

How Are Pages Rendered?

Like this:

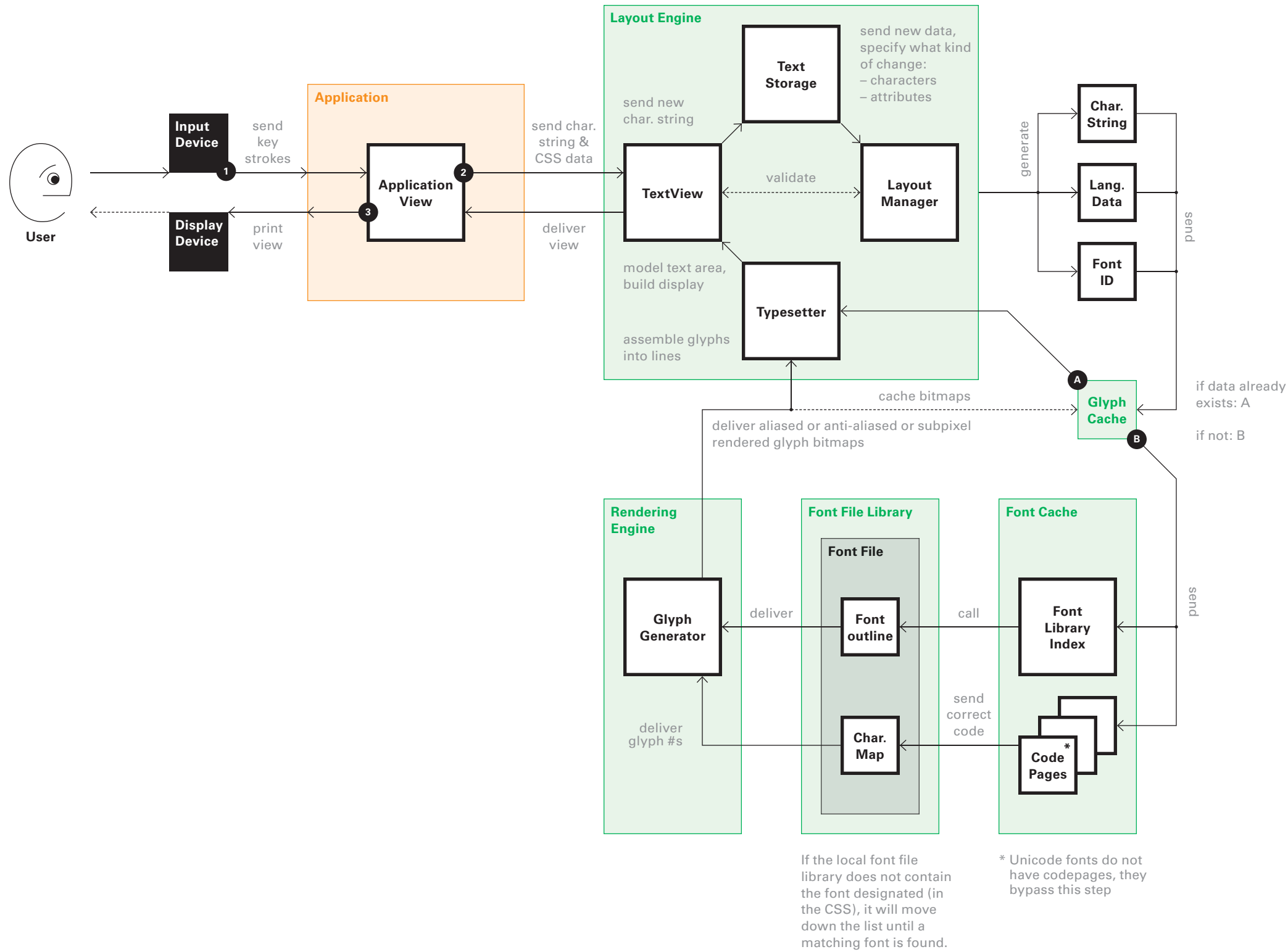
- Open Document
- Determine language
- Find plane
- Determine font
- Find font
- Determine size
- Render font at size
- Cache glyphs
- Match text to appropriate glyphs
 - (some languages require “shaping”)
- Find line lengths and leading
- Place first glyph
- – Check kerning table for special spacing instructions
- Place next glyph
- IF: current position > line length THEN: move to next line
- ELSE:

With H&J settings,
this process becomes much more complicated

How Are Pages Rendered?

Hello, World!

What happens when you type in a text editor?
This example is based on Mac OS.



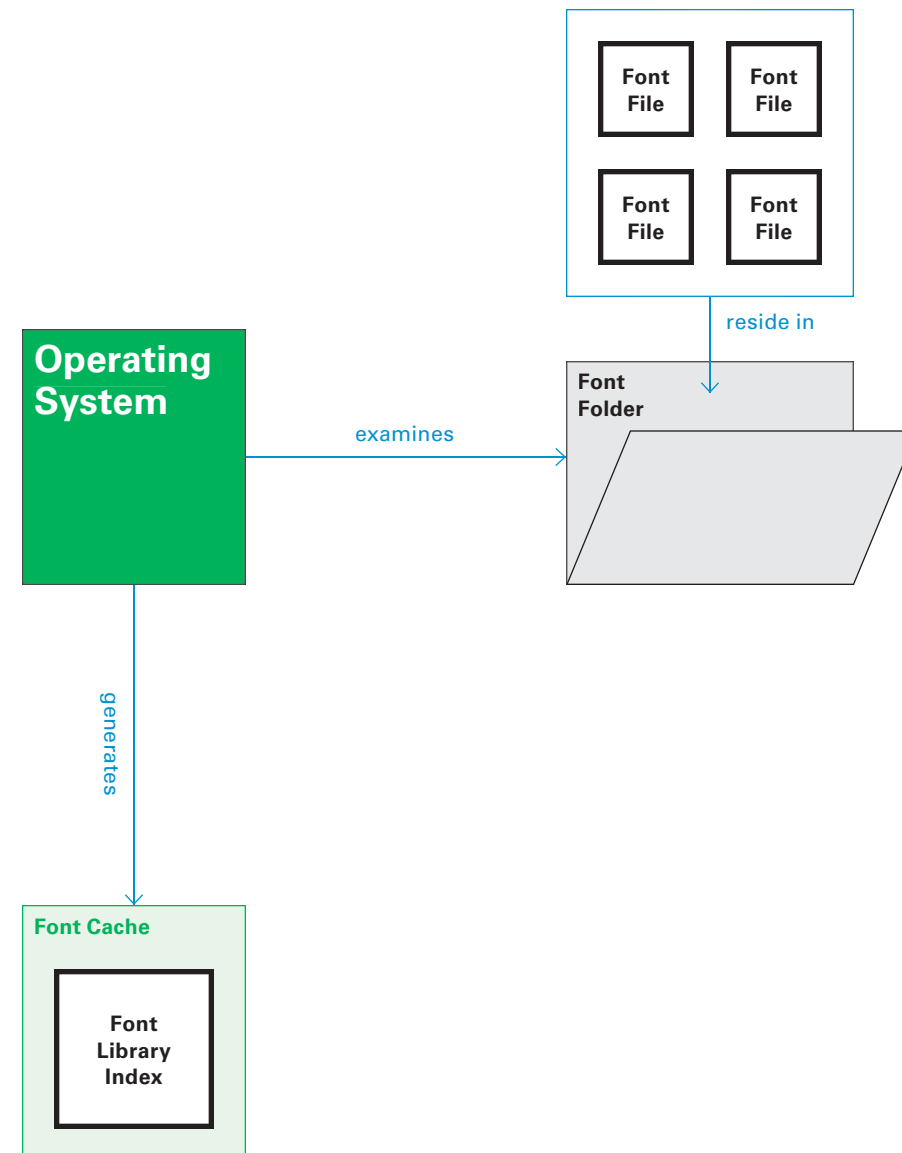
How Are Pages Rendered?

Font Cache

Fonts are digital files that must be accessed by the operating system in order to generate glyph bitmaps. Simply having a font file on your computer's hard drive does not allow it to be accessed. The operating system must first index the font file into the font cache. Caching is the process of saving something that is likely to be reused – thus saving time.*

The font cache is a list of fonts accessible to the system. This terminology is potentially confusing because the font cache is *not* all the fonts stored on the device; it is the list of fonts known to the operating system. Most operating systems have one or more designated font folders. Any font file placed in those folders will be indexed into the font cache while font files that are not will remain invisible to the operating system.

* This time saved is achieved at an expense: the cached data takes up memory that then can't be used to store other data. Caching is also “expensive” (in terms of demands on the OS) because work must be done to ensure that the cached data is current, that the computation that generated the need for the data is current, and the arguments requiring the computation are current. Additionally, this process makes the computation more complex, which can lead to difficulties when these processes need altering or when the computation fails part of the way through.

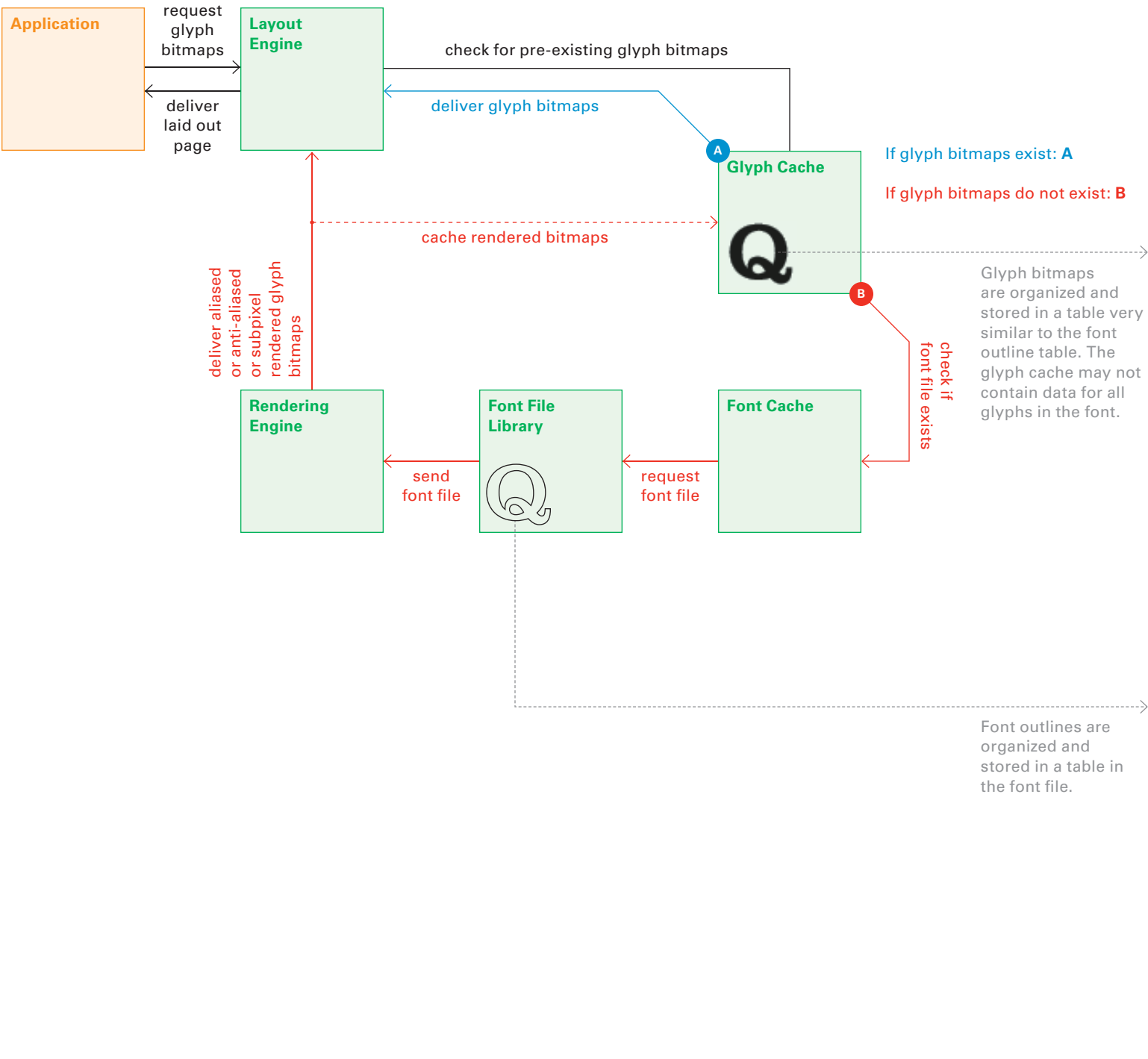


How Are Pages Rendered?

Glyph Cache

When an application asks the operating system to render glyph bitmaps, the operating system first checks the glyph cache to see if those bitmaps already exist; if they do, the OS delivers them to the application. If they do not, the OS renders them first and then delivers them to the application. **Glyph caching is a strategy for conserving resources and avoiding repeating tasks when possible.**

It is important to note that font caching and glyph caching are distinct operations.



How Are Pages Rendered?

Contextual Shaping

To compose a line, the layout engine must request glyph bitmaps from the rendering engine. When laying out lines of text that use contextual shaping, the layout engine must first determine what version of the character to ask for (i.e. what glyph). Before sending the request to the rendering engine, it first checks what position in the word the character occupies and what characters surround it. In most Latin scripts this process is mainly used to determine when to use a ligature. In other scripts, such as Arabic or Indic, this process is extremely important; without it, the language could not be displayed properly. Contextually orientated glyph usage isn't an aesthetic choice, as in English, but a grammatical requirement. (See page 62 of *Understanding Typography*.)

Contextual shaping can be a very demanding activity for the layout engine, requiring multiple times the amount of system resources that simple text layout does. Some application layout engines, such as the one in WebKit, have to outsource line layout to the OS when there is contextual shaping because it is too demanding. Because of this, contextual shaping is turned off for most Latin-based scripts by default.

Isolated Letters

7: Ta Marbutah

6: Ya

5: Ba

(space)

4: Ra

3: Ayn

2: Lam

1: Alif

العربية

← Reading direction

Contextual Versions

Final

Medial

Initial

(space)

Final

Medial

Initial

Initial

العربية

Assembled Word

Latin transcription: Al-'Arabiyya

Translation: "The Arabic"

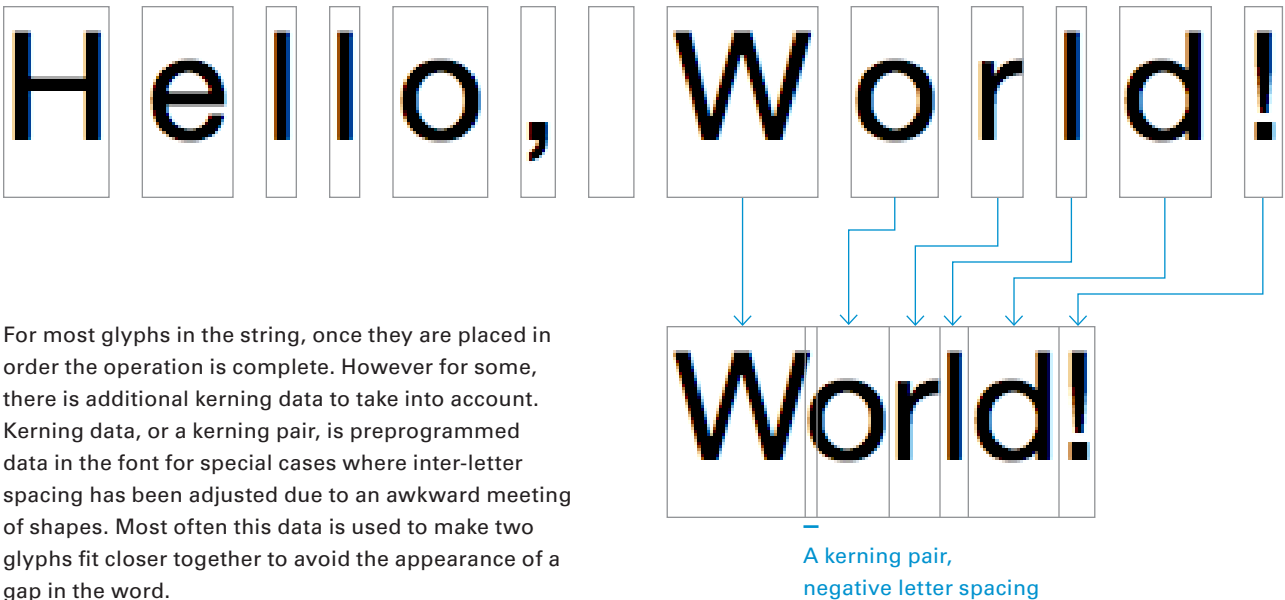
العربية

Sources:
msdn.microsoft.com/en-us/goglobal/bb688172
people.w3.org/rishida/scripts/featurelist/index.php?col=6

Line & Paragraph Assembly

After glyph bitmaps have been generated from the outline font file, they need to be composed into a page. The first step in this process is line assembly. **The layout engine receives data about the start location and the length of the text line – it then begins placing glyphs side by side until it reaches the end of the line.** When the end of the line is reached, the layout engine will attempt to place any remaining glyphs on the next line. Doing this sometimes requires that a hyphen be added or a glyph be substituted (e.g. if a ligature falls at the end of a line and needs to be broken for hyphenation).

Individual glyph bitmaps are arranged sequentially based on the character string. The data sent to the layout engine includes the bitmap image of the glyph along with the size of its “side bearing” (the amount of space to the left and right of each glyph).



The final line is assembled.

Hello, World!

H&J

- Number of hyphens in row: Should not exceed two in a row.
- Space between words: Should be between 1/8 and 1/2 of an em quad.
- Space between letters: Should be between 75% and 125% of the default sidebearings as set in the font file by the designer.

Hyphenation & Justification Algorithm



How Are Pages Rendered?

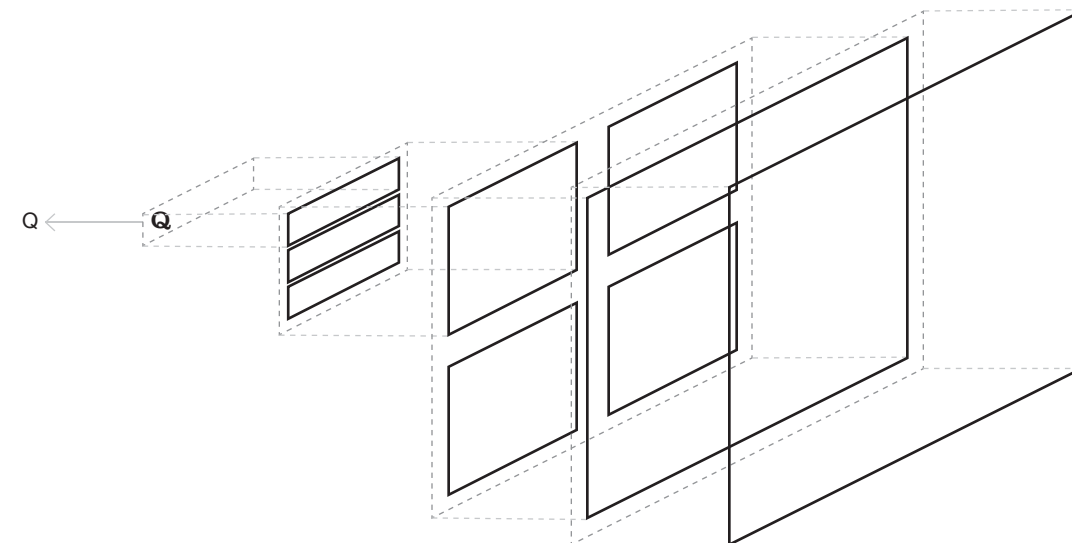
Page Assembly

Once lines and paragraphs of text have been composed, they must be arranged on the page. This level of composition is **essentially the placing of rectangles next to or inside other rectangles**. In the following hierarchy:

- Application
- Window
- View
- Container
- Object

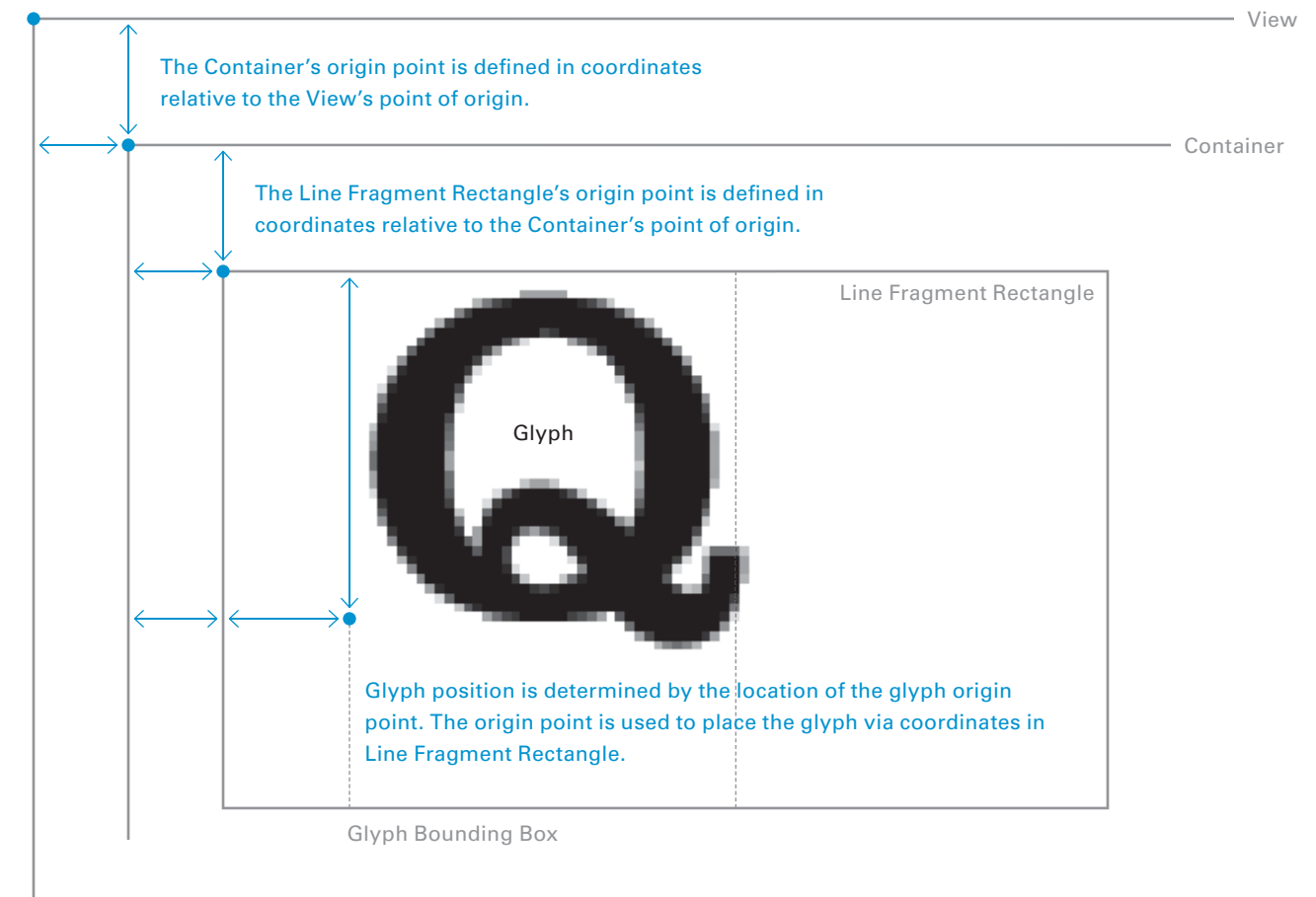
The arrangement of various page components is handled by the layout engine.

In many programs this task is delegated to the layout engine of the operating system; however a number of applications such as Web browsers use their own embedded layout engines.



Character Glyph Line Fragment Rectangles Text Containers Application View Application Window

The View origin point is typically the zero point of the application window's non-navigational space, e.g. the upper left corner of a Web browser window, below all application navigation.

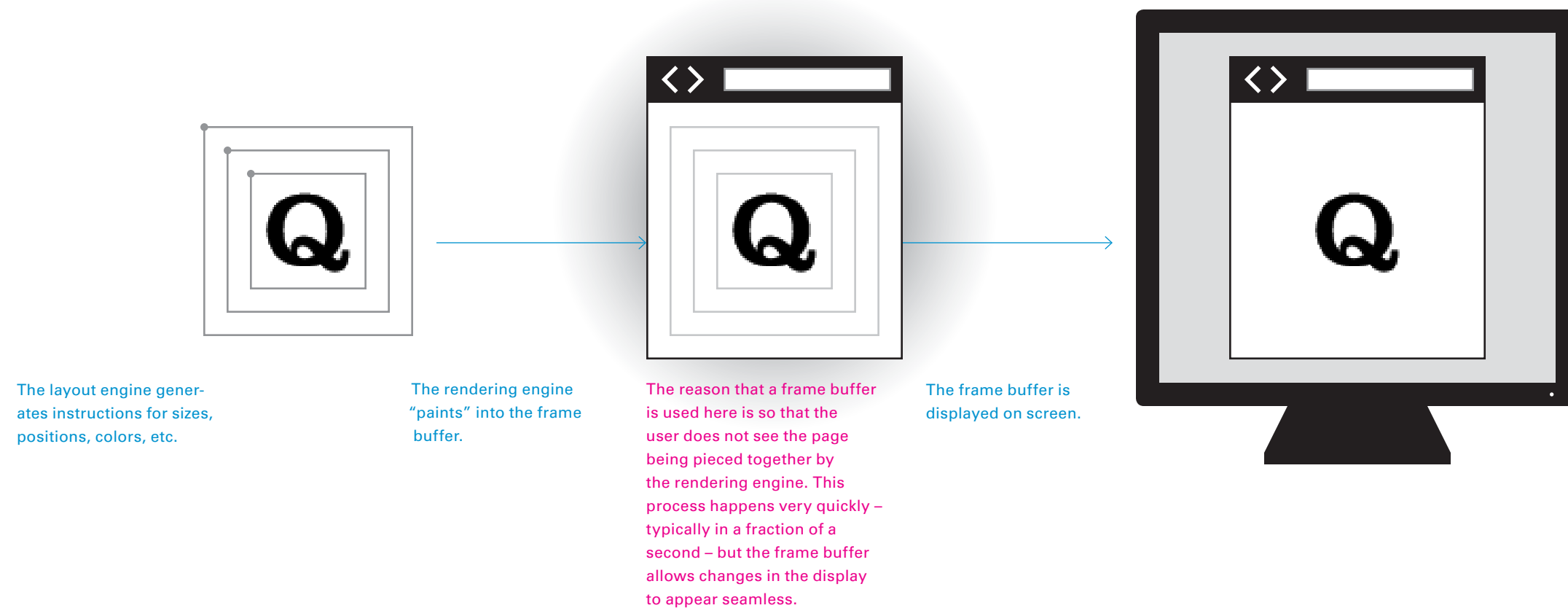


How Are Pages Rendered?

Frame Buffer

A “buffer” contains data that is stored for a short amount of time. The portion of computer memory reserved for holding the complete bit-mapped page image that is sent to the monitor is called the “frame buffer”. The frame buffer can be stored in either the general main memory (RAM) or specialized video memory (VRAM) on the video adapter.

The term “frame buffer” can be a confusing one – it is sometimes used to refer to a physical piece of computer hardware that stores display data in which case the term “screen buffer” is used to refer to the actual data.



What Is a Digital Book?

Access

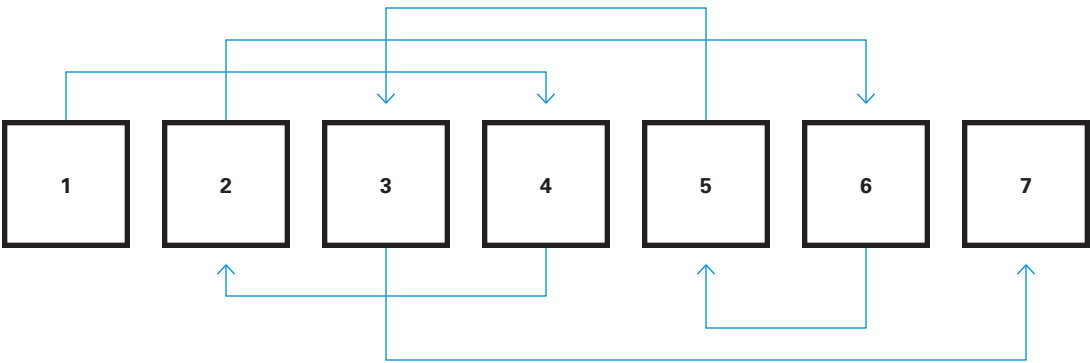
Content in digital books can be accessed in two ways: **random access** and **serial access**. Random access is when a group of elements can be accessed in an **arbitrary sequence with no predetermined order in equal time** (e.g. a card catalog). Sequential access is when a group of elements is accessed in a **predetermined, ordered sequence** (e.g. a scroll)

Users want to know:

- Where am I?
- Where have I been? ← History
- Where can I go?

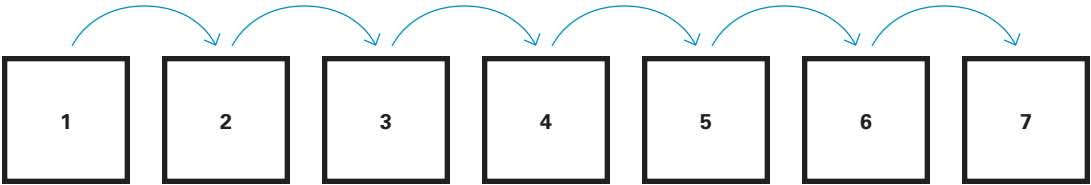


Random Access (also sometimes called “direct access”)
In a data structure, random access implies the ability to access any element in the same amount of time – e.g. moving from 1 to 5 takes the same amount of time as moving from 1 to 2.



It is not necessary to represent random access content as a sequence of elements in a row. Random access content can also be represented as a database with content in no particular order – multiple ordering schemes are possible, and any ordering scheme chosen by the author is arbitrary.

Serial Access
In a data structure, content is said to be sequential if a user can only visit the elements in one particular order – e.g. moving from 1 to 2 takes half the time it takes to move from 1 to 3.



What Is a Digital Book?

Hypertext

Digital texts differ from analog texts in the ease with which content can be connected for navigation and referral – **hypertext provides a way for users to move through or to a new text quickly**. Hypertext is built on the idea of “links”, and originally there were many kinds. However today links are only used in two primary ways: to navigate within a document through “anchors” and to navigate to a new document through a “referral” – both of these kinds of links are one-way.

The term hypertext was coined by Ted Nelson in 1965, though the idea was first proposed in 1945 by Vannevar Bush (see next page). Nelson developed a model for creating and using linked documents and later developed the Hypertext Editing System (in 1967). Douglas Englebart also developed a similar system, called NLS (oN-Line System), in 1962.

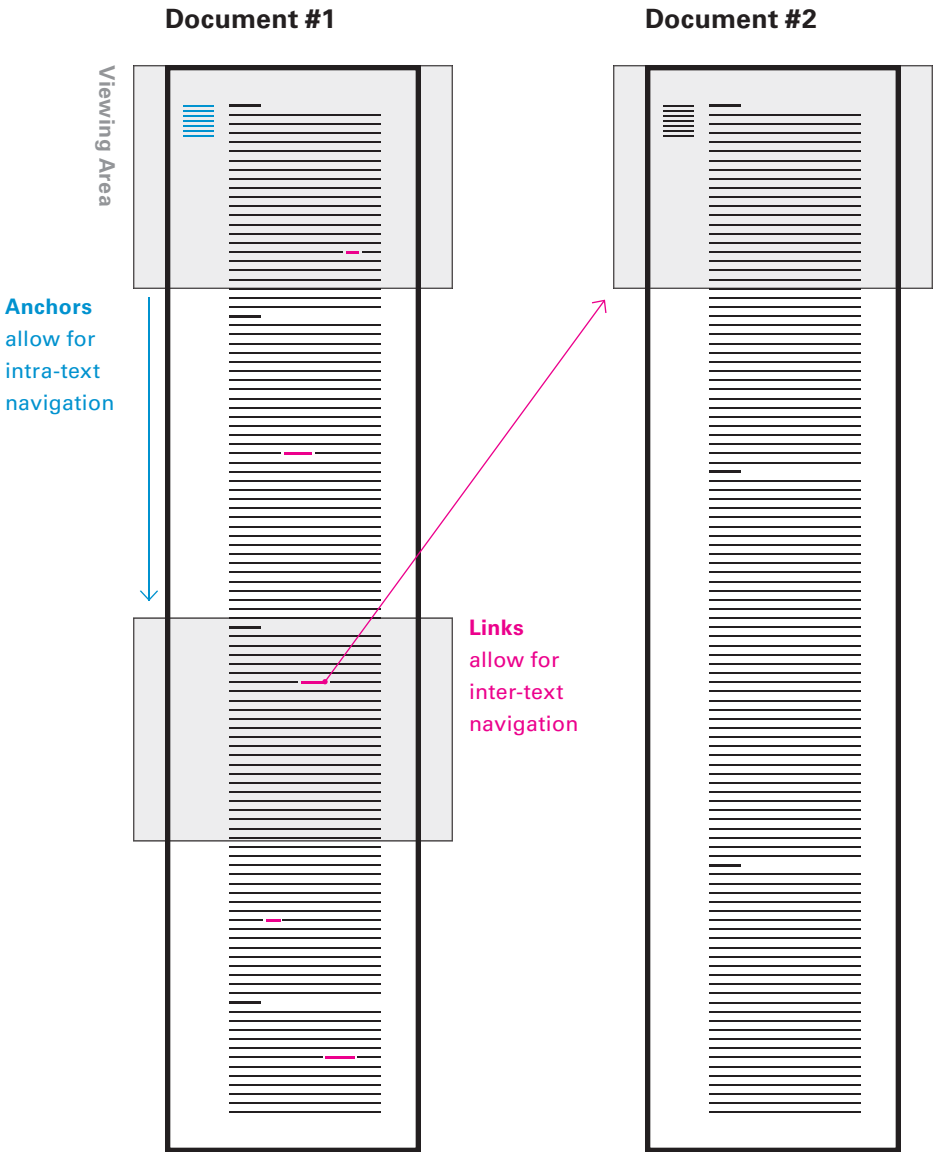
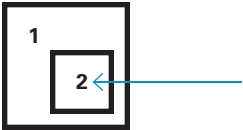
In the contemporary HTML specification, **hyperlinks are uni-directional and one-to-one**; however there is no reason that a hyperlink couldn’t be one-to-many (i.e. clicking a single link opens multiple web-pages). The ENQUIRE system, on which HTML was based, featured bi-directional links; however it became apparent that the system was not scalable because bi-directional links require there be a system administrator who ensures that all links point in both directions (i.e. anytime a page is created with a link to a separate page, if the link is to be bi-directional, the linked to page has to be updated as well).

Most links today are premised on the idea of the user going somewhere – the user clicks a link and the browser loads the page that the link pointed at. There are also links that bring the linked content to the user – frames were often used to accomplish this, allowing one page to load inside of another page.

You go there



It comes here



Original HTML Link Relationship Values

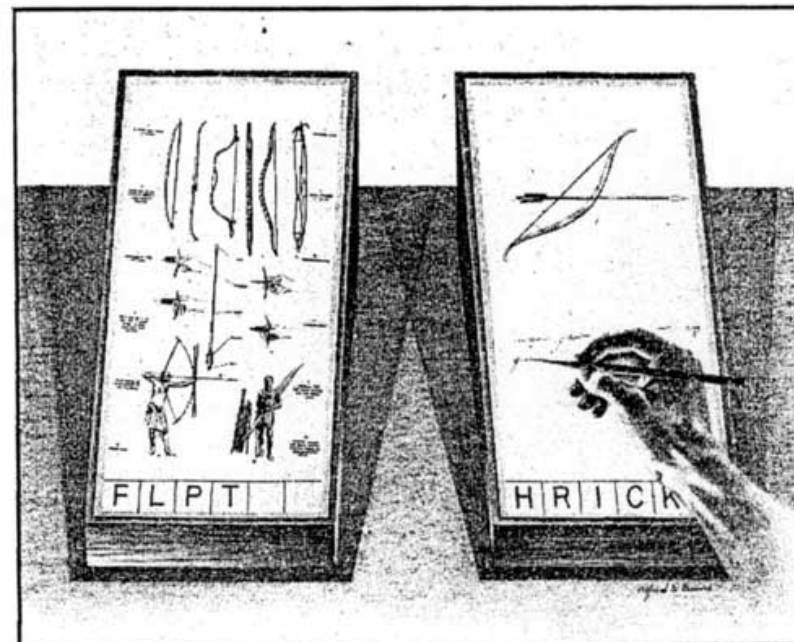
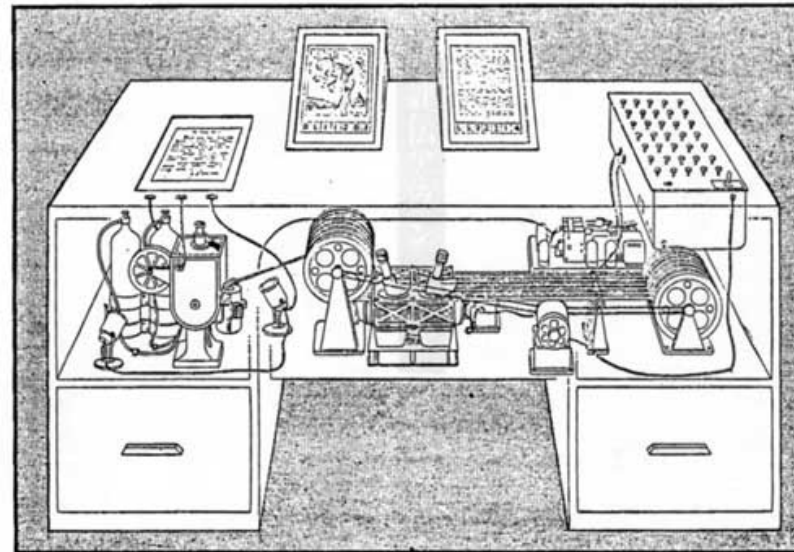
Relationships Between Documents	
Use Index	Document A uses document B as an index.
Use Glossary	Document A uses document B as a glossary.
Annotation	Information in document B is additional and subsidiary to A, used by author of B to write the equivalent of a margin note on A.
Reply	Similar to Annotation, but B is not subsidiary to A.
Embed	If link is followed, B is embedded into A.
Precedes	In an ordered structure, A precedes B, B follows by A. Any document can only have one link of this type.
Subdocument	B is a lower part in a hierarchical relationship to A.
Present	Whenever A is presented, B must also be presented.
Search	When link is followed, B should be searched rather than presented.
Supersedes	B is a previous version of A.
History	B is a list of versions of A.
Relationships About Subjects to Documents	
Includes	A includes B, B is part of A.
Made	Person (etc.) described by A is author of B.
Interested	Person (etc.) described by A is interested in B.

What Is a Digital Book?

Memex

In 1945 Vannevar Bush published an article titled *As We May Think* that predicted many kinds of technology including hypertext, personal computers, the Internet and World Wide Web, speech recognition software, and online encyclopedias such as Wikipedia. One of the propositions of the article was a device called a “memex” (a portmanteau of “memory” and “index”), a theoretical proto-hypertext computer system in which an individual could compress and store all of their books, records, and communications. In the memex, documents can be edited, annotated, and linked together – the device was intended to be a supplement to an individual’s memory.

The memex is conceptually related to the “commonplace book”. Commonplace books were notebooks into which people entered, by hand, information they collected around a theme. A commonplace book was an essential personal tool for anyone doing research into a topic during times when books, much less libraries, were rare.



Conceptual illustrations of a memex. The device was conceived as a desk-like apparatus, with a platen scanner for input of documents and screens for viewing and editing more than one document at a time.

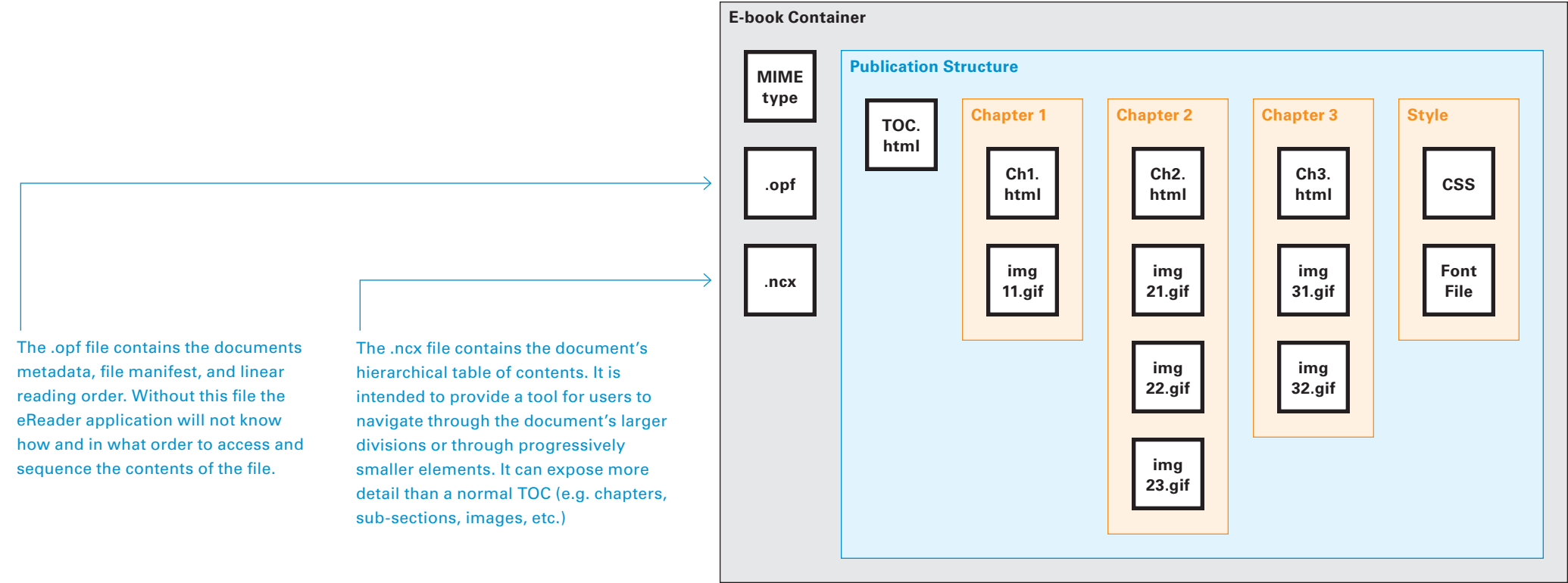
What Is a Digital Book?

E-book Formats

Almost all e-book formats are, at their core, HTML. HTML documents have a size limit beyond which they become very slow and may potentially crash the reader application. That’s why e-books typically break a book into chapter documents and package them into a hierarchical file structure along with any images and resources (such as fonts) the e-book may need.

Generic E-book Format

E-books come in many formats, but the majority of them use a format similar to what is shown below. Each chapter is an independent folder with all the files that comprise that chapter. Chapter folders, style data, and a table of contents (TOC) file are packaged together. This package is wrapped in a container that handles compression, encoding, and permissions.



How Are Digital Books Managed?

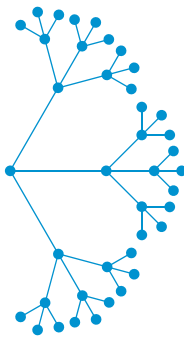
How Are Digital Books Managed?

Directories

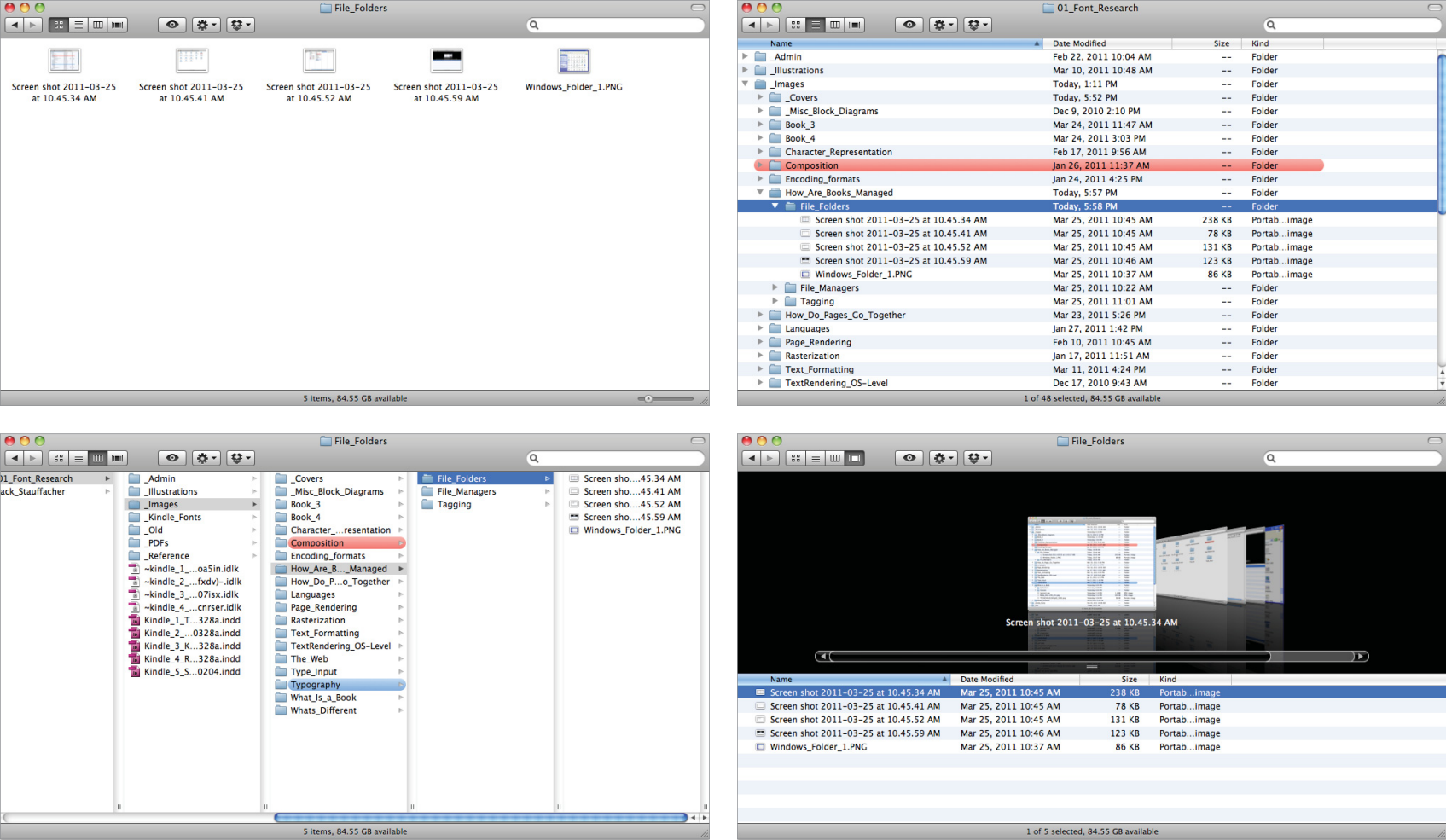
At the most basic level, files are managed in most OS interfaces through the use of directories. In contemporary operating systems, the primary metaphor is that of a file folder. This system is relatively simple and easy to comprehend but can be limiting. Files can be in only one directory at a time. Although aliases to a file can be placed anywhere, this is a clumsy way to handle increasingly hybrid data.

No taxonomy can ever be 100% definitive, and it is always possible to organize material in multiple ways: a pile of coins can be organized by their monetary value, or by their circumference, or by their thickness, or by their color, or the images they bear, or by the number of letters they have, etc. The fact that there is no one “right” way to organize any set of items becomes even more problematic in the context of digital information.

The directory model of content organization has a conceptual limit, i.e. there is a limit to the number of items in a tree that a user can remember. Because of this, search tools become extremely important.



Directories are tree structures. There is a root directory, in which there can be any number of other directories, in which there can be any number of other directories, and so on. With limited numbers of files, this model of organization is easy to use and simple to navigate. The drawback of this is that with larger numbers of files (and directories), it can become difficult to locate a single file.



The Mac OS Finder allows users to view their file directory structure in several ways, but always through the metaphor of a file folder.

The URL (Uniform Resource Locator) for the above images:

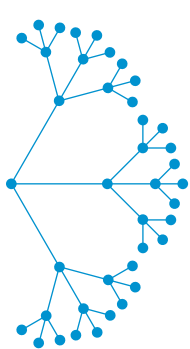
C://HD/Users/User _ Name/Documents/Work/Kindle/01 _ Font _ Research/_ Images/How _ Are _ Books _ Managed/File _ Folders/

How Are Digital Books Managed?

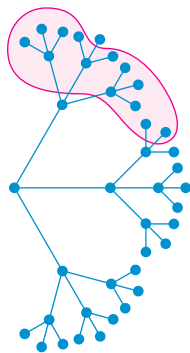
Tagging

In contrast to the location-based organizational method of directories, tags allow users to organize files into virtual collections. A single file can be have many tags, thus allowing it to be part of many virtual collections. In addition, tags can be nested and have a hierarchy. Tags enable users to filter search results through inclusion and exclusion of tags.

The primary difficulty of tagging is that it is mostly manual – someone has to add the tags, a far more time consuming process than putting a file in a directory. It is possible to imagine a system that simplified this process through the use of inherited tags, but as of today such systems have not seen widespread use. Social tagging systems such as Seadragon, MusicBrainz, and LibraryThing offer promise.

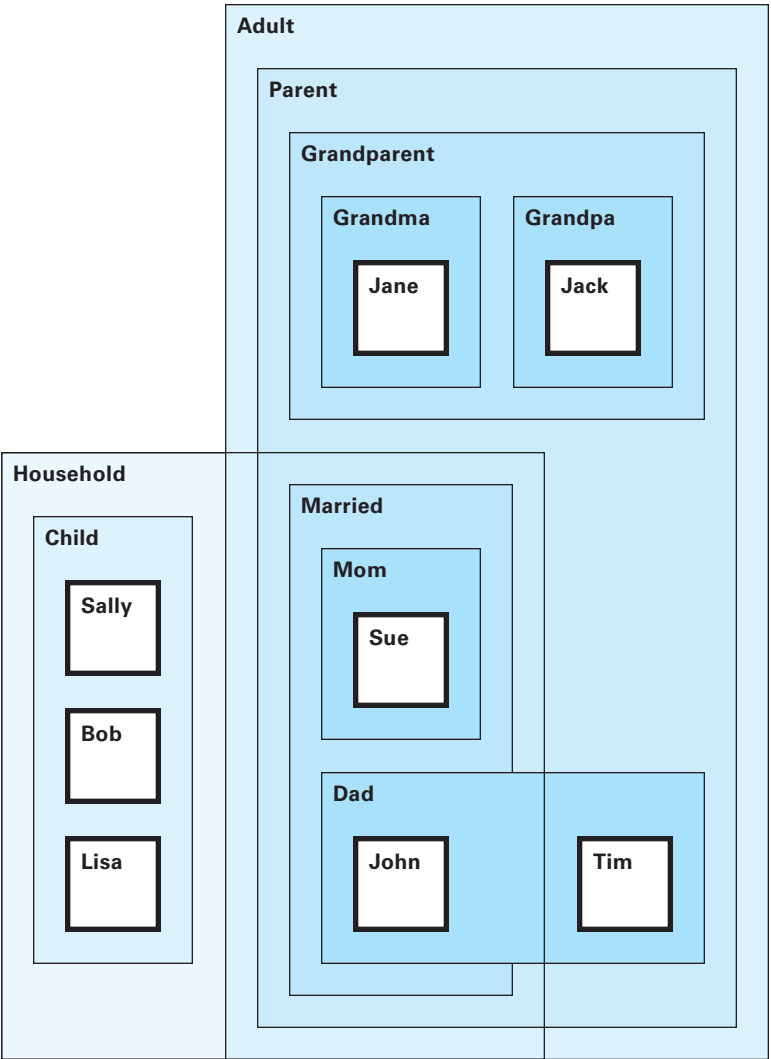


Directory Structure



Nested Tags
Like directories, tags allow for nested hierarchy. Unlike directories, tags also allow for items to be grouped outside of a linear tree structure – items can be in a hierarchical branch of a tree *and* in a group that crosses tree branches.

Family Structure



The advantages of tagging over directories become apparent when attempting to classify something as part of more than one collection. Tag-based categorization can be more specific and detailed.

Resulting Tags

Sally

Sally is
a Child,
and part of the Household
but not an Adult.

John

John is
a Dad,
Married,
a Parent,
but not a Grandparent,
an Adult,
a part of the Household,
but not a Child

Tim

Tim is
a Dad,
but not Married
a Parent,
but not a Grandparent
an Adult,
*but not part of the Household,
or a Child.*

Nested sets are very useful when trying to fine tune a search. Using directories alone could not achieve the same level of detail and specificity as tagging in this case – for instance, in the following directory structure, there is no way to efficiently categorize Tim or have people who are Adults and not part of the Household:

- Household
 - Child
 - Parent
 - Married
 - Dad

Directories do not allow cross-listing or exclusions.

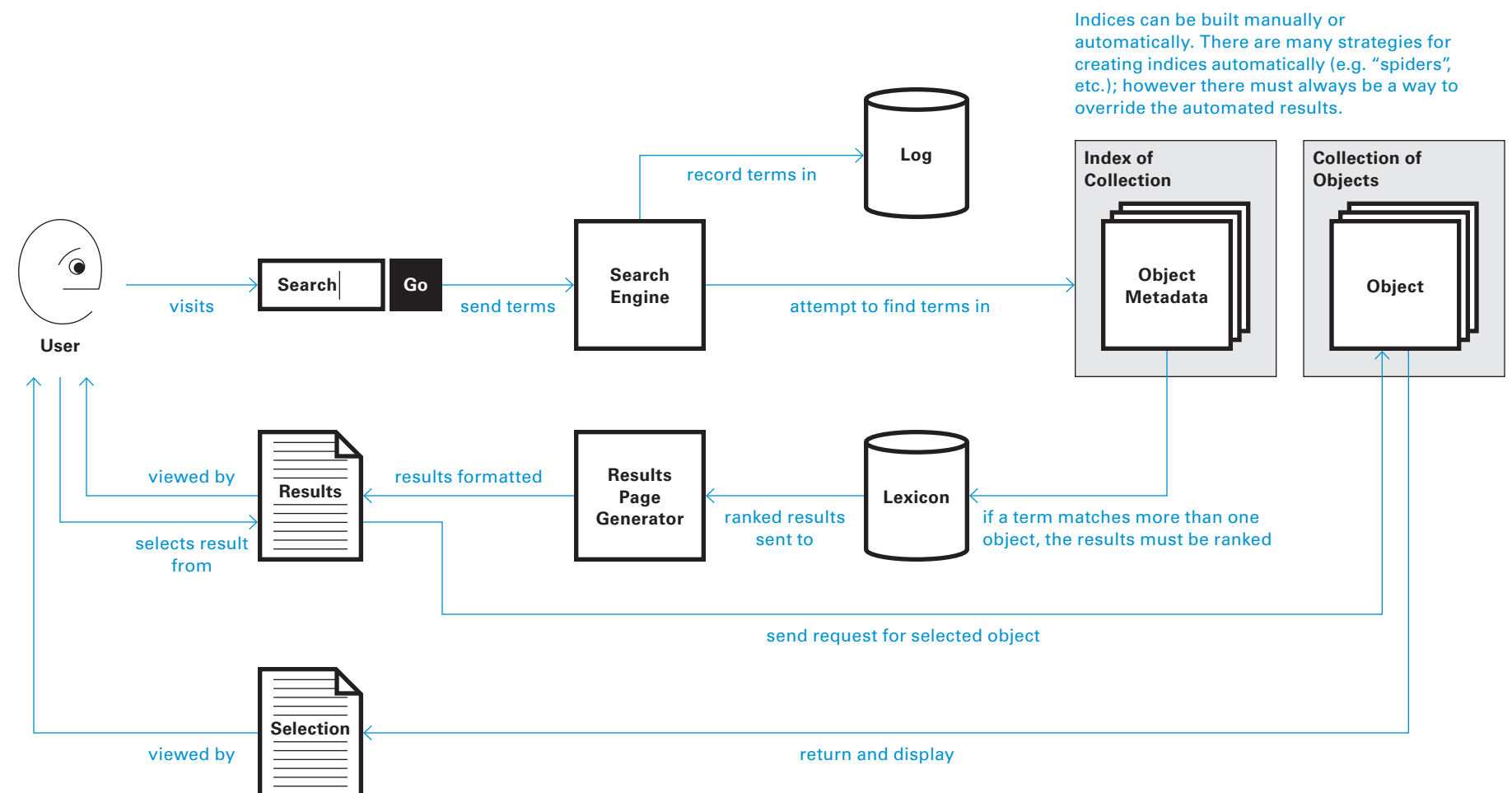
Search

A user of a search service enters terms in a form. Clicking on an action button in the search form sends the request to the search engine. **The search engine compares the request with entries in an index.** In the index, each entry is associated with one or more pointers (usually URLs). If a requested term matches an entry, the search engine returns the associated pointers in the form of search results.

If the search engine finds multiple pointers associated with a requested term, it ranks them according to various algorithms. It may also consult a lexicon file which forces some pointers to the top of the list. The rules by which a service ranks results are often unavailable to users.

Depending on the search service, users may constrain their searches in several ways including:

- by using Boolean operators to add or exclude terms
- by limiting the range: collection, domain, or language
- by specifying date, author, region of origin, or other metadata
- by specifying media type



How Are Digital Books Managed?

Library Management Tools

There are many kinds of applications that can be used to organize and manage a digital library. Some simply provide a method for adding tags to files on a hard drive (e.g. Leap) while others tag and store data inside a proprietary database (e.g. Yojimbo). File management applications tend to fall into two categories: generalist managers (“anything buckets”) for almost any file type and specialist applications designed to handle one specific file type. **Generalist managers offer extended functionality for sorting files beyond the OS, while specialist managers also usually provide tailored viewing options and more detailed metadata entry.** Most library management tools support both folders *and* tagging; however both types typically prevent users from accessing data managed by the application outside the application.

Some popular library management tools are:

- Books
- Daneismos
- Delicious Library
- Endnote
- Evernote
- Leap
- Papers
- Together
- Yojimbo

Together is a generalist application for cataloging your digital libraries. The application allows users to sort and categorize the material in their library, but the viewing options are not as powerful as specialist applications made for PDFs and e-books (see below).



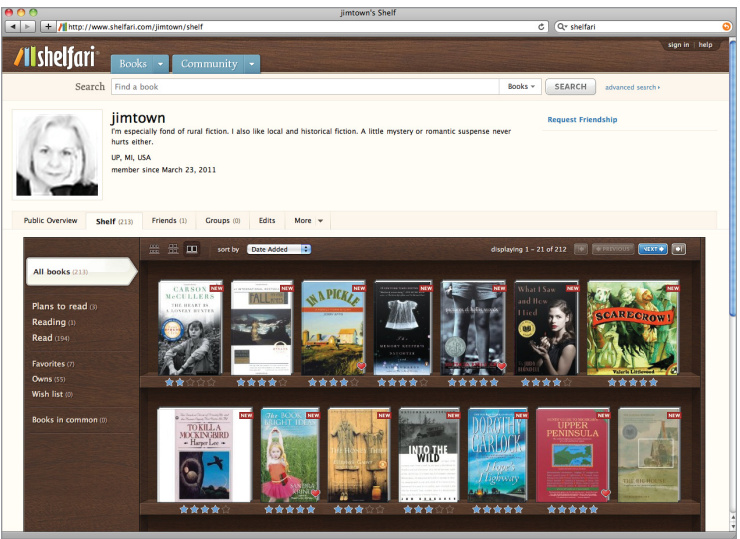
Papers is an application for managing PDFs. Beyond sorting and tagging, the application provides users with a way to enter full bibliographic data to each file and a multitude of viewing options.



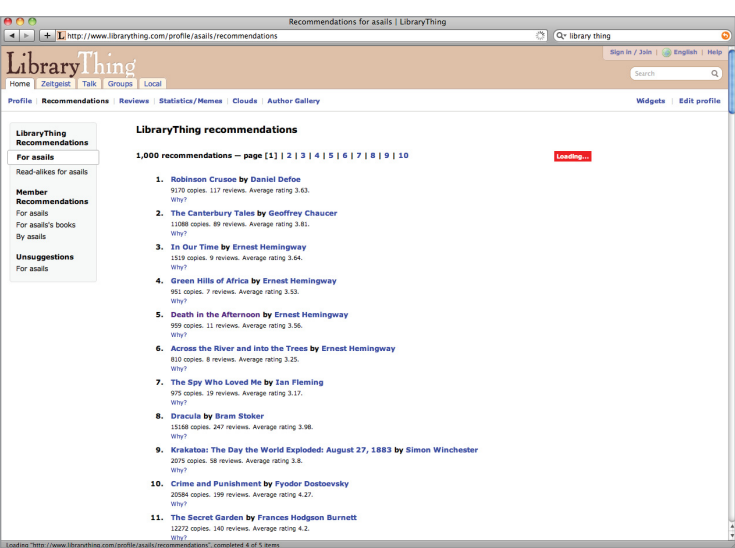
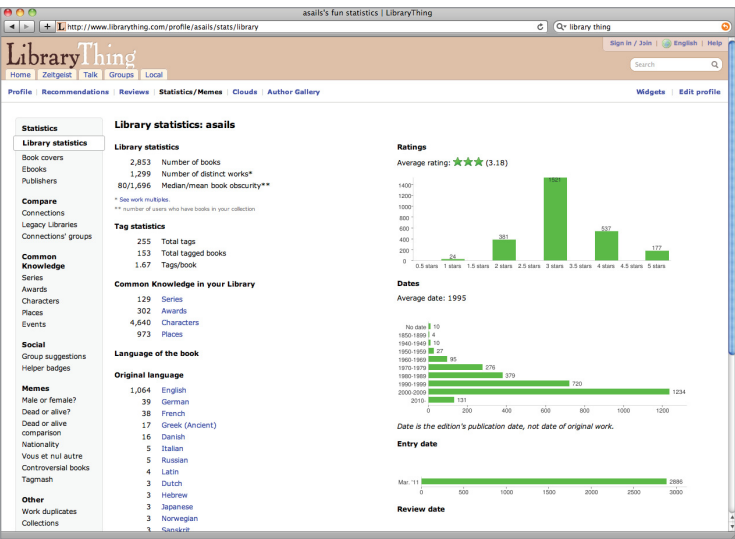
How Are Digital Books Managed?

Online Library Management Tools

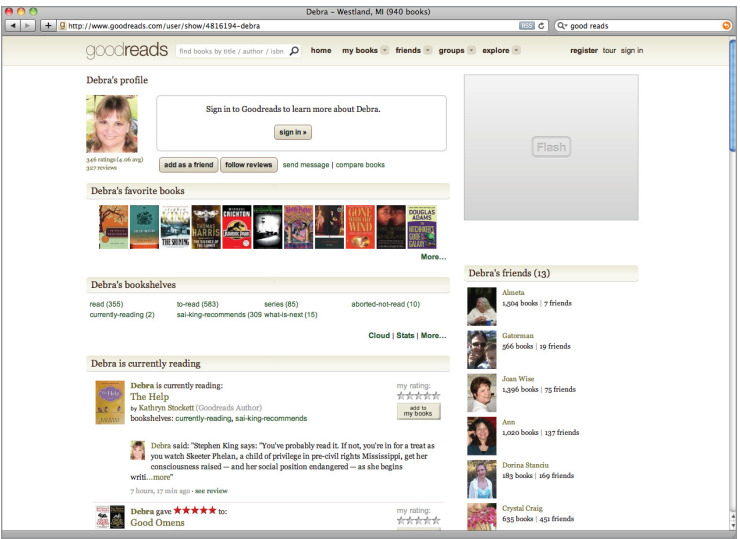
In recent years, file management tools have begun migrating online. Most online tools for file management do not hold the actual files, but rather simply provide users with a tool for cataloging their offline libraries.



Shelfari is a social media site and “community-powered encyclopedia” for books.



Library Thing is a website for cataloging an offline library – the contents of the library are analyzed by the site to provide users with statistics about their material and recommendations for further reading.

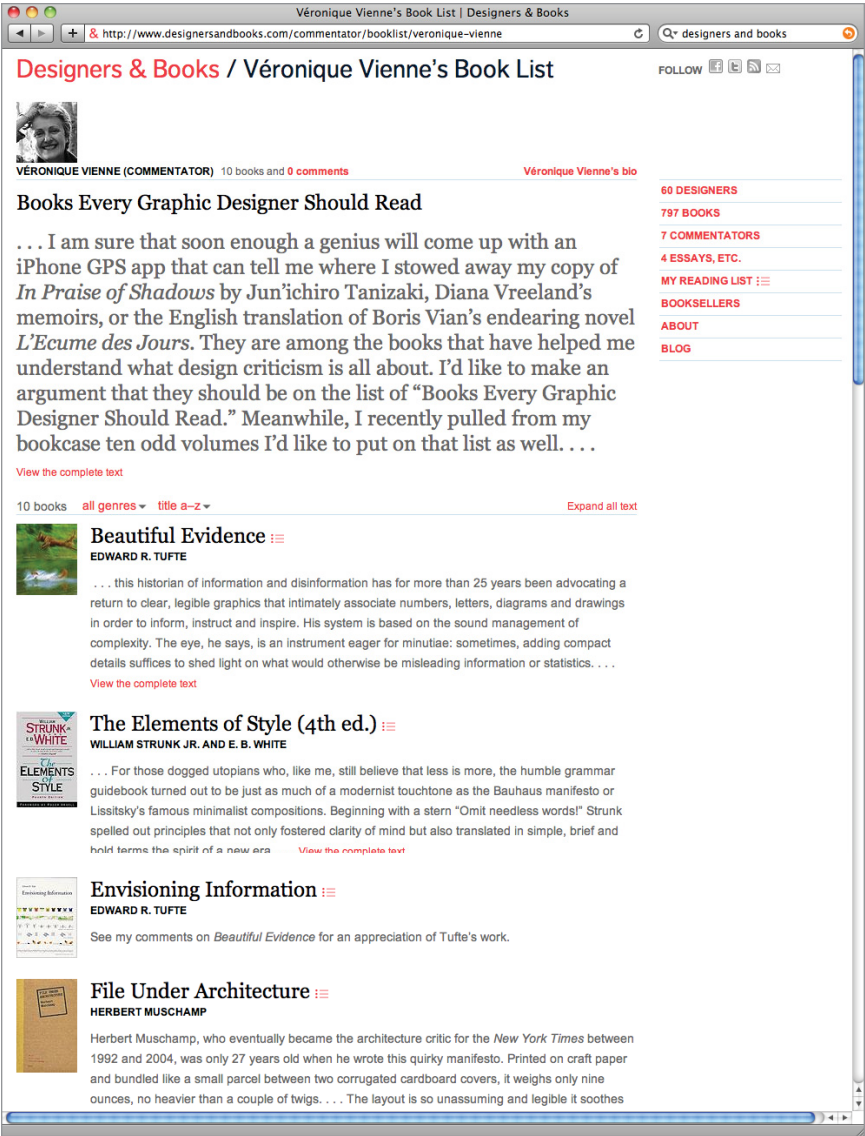


Good Reads lets user catalog their libraries and share their impressions with others.

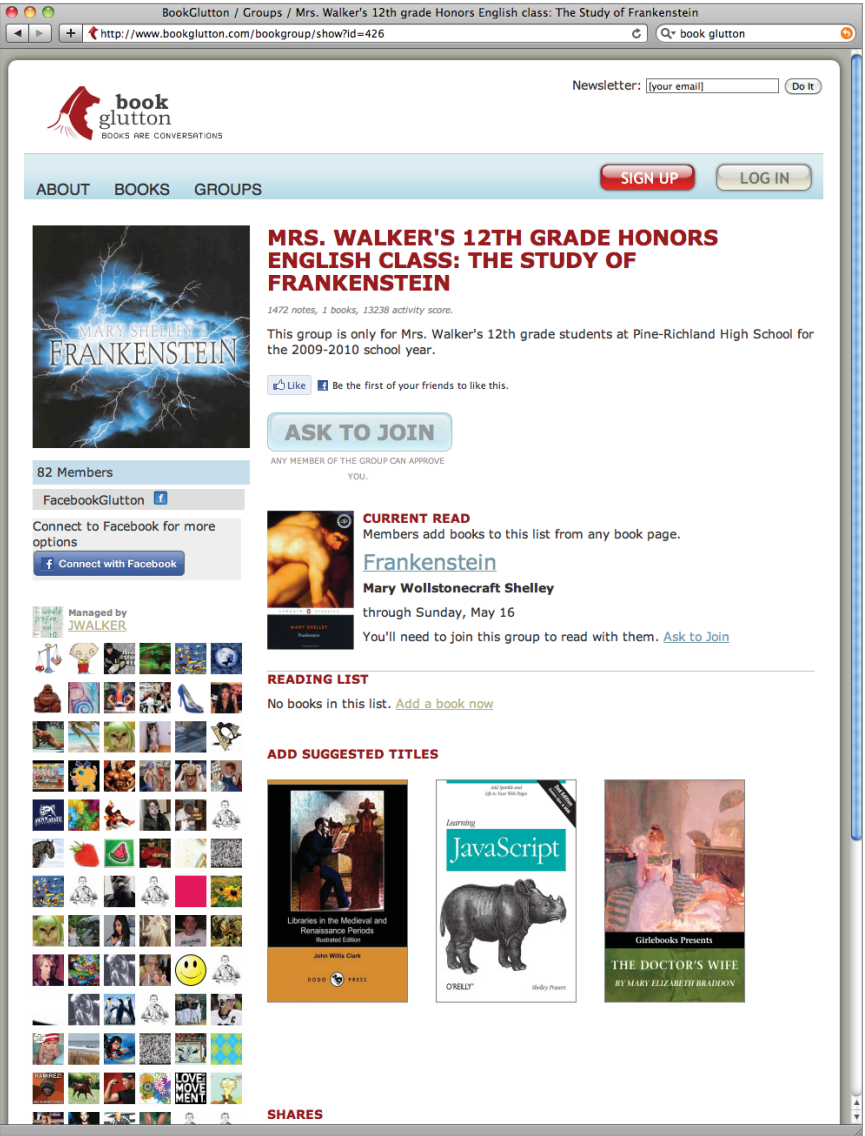
How Are Digital Books Managed?

Online Social Book Services

Social services for sharing the reading experience are emerging online – digital book groups that allow users to discuss texts.



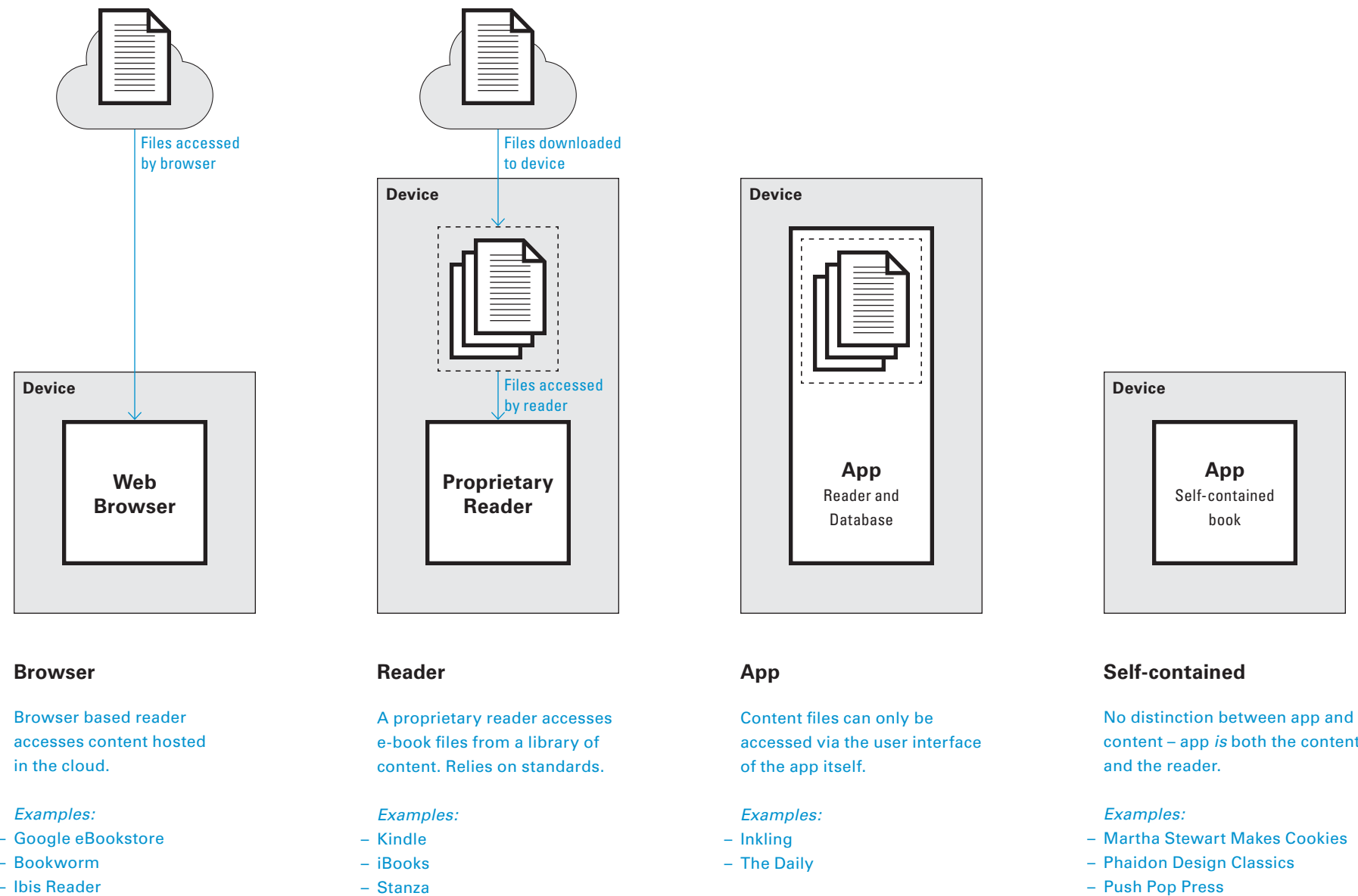
Designers and Books is a site where invited designers can review books and share their recommended book lists.



Sites such as Book Glutton allow users to share their reading list and discuss books with other members of their reading group.

Types of E-book Readers

E-books are delivered, consumed, and managed in a number of ways ranging from in-browser online reading with content hosted in the cloud to self-contained apps that are both reader application and content.



Appendix:

How Do Fonts Work on the Web?

The Web requires a special class of font usage and display technologies and techniques.

Year	Event
1995	 tag introduced by Netscape
	–face attribute of introduced by Microsoft (with Internet Explorer 1.0)
1996	CSS1 introduced allows for selection of typeface through style sheet
	Embedded Open Type (.eot) introduced by Microsoft (with Internet Explorer 3.1)
1998	 deprecated, HTML 4.01
	@font-face proposed, CSS2
	@font-face removed, CSS2.1
2005	@font-face fully implemented, CSS3

Browser Layout Engines

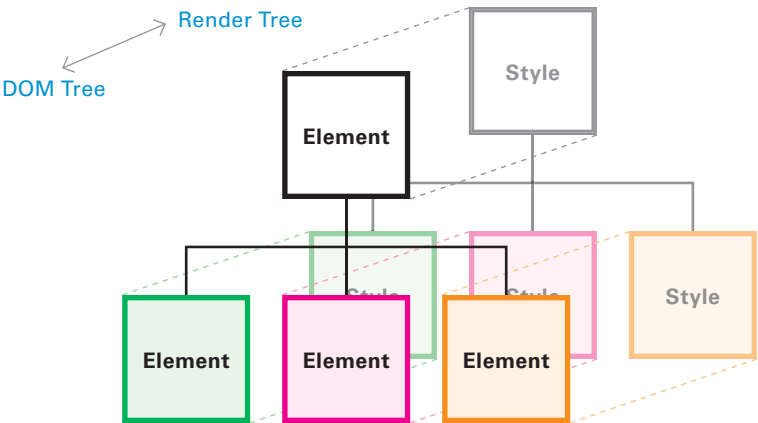
A Web browser layout engine (sometimes confusingly referred to as a rendering engine), is a software component that **takes marked up content** such as HTML, XML, image files, etc. and style information such as CSS, XSL, etc. and **displays formatted content on screen**. A Web browser layout engine can also be embedded in e-mail clients, on-line help systems, or other applications that need to display and allow editing of Web content.

		Gecko*	Presto	Trident	WebKit*
Developer		Mozilla Foundation	Opera	Microsoft	Apple, KDE, Nokia, Google, RIM, Palm, & others
Used in		Firefox, Camino, SeaMonkey, Galeon, K-Meleon, Flock, GNU IceCat, Icedove etc.	Opera, Opera Mobile, Nintendo DS browser, Internet Channel	Internet Explorer	Chrome, Safari, AIR, Android, Symbian S60, OmniWeb, Palm webOS, Adobe Dreamweaver CS4
Version support for CSS3 font resources	@font-face	1.9.1	2.2	<3.1 [IE 4.0]	525
	src	1.9.1	2.2	Partial	525
	font-stretch	-	-	5.0	-
	unicode-range	-	-	5.0	Partial
	font-variant	-	-	-	-
	font-feature-settings	-	-	-	-
Version support in HTML & XML documents (for fonts applied in @font-face rule)	Embedded OpenType	-	-	<3.1 [IE 4.0]	-
	TrueType	1.9.1	2.2	5.0	525
	OpenType	1.9.1	2.2	5.0	525
	SVG	-	2.2	-	525
	Web Open Font Format	1.9.2	-	5.0	533

* Both Gecko and Webkit are open source

Webpage Rendering

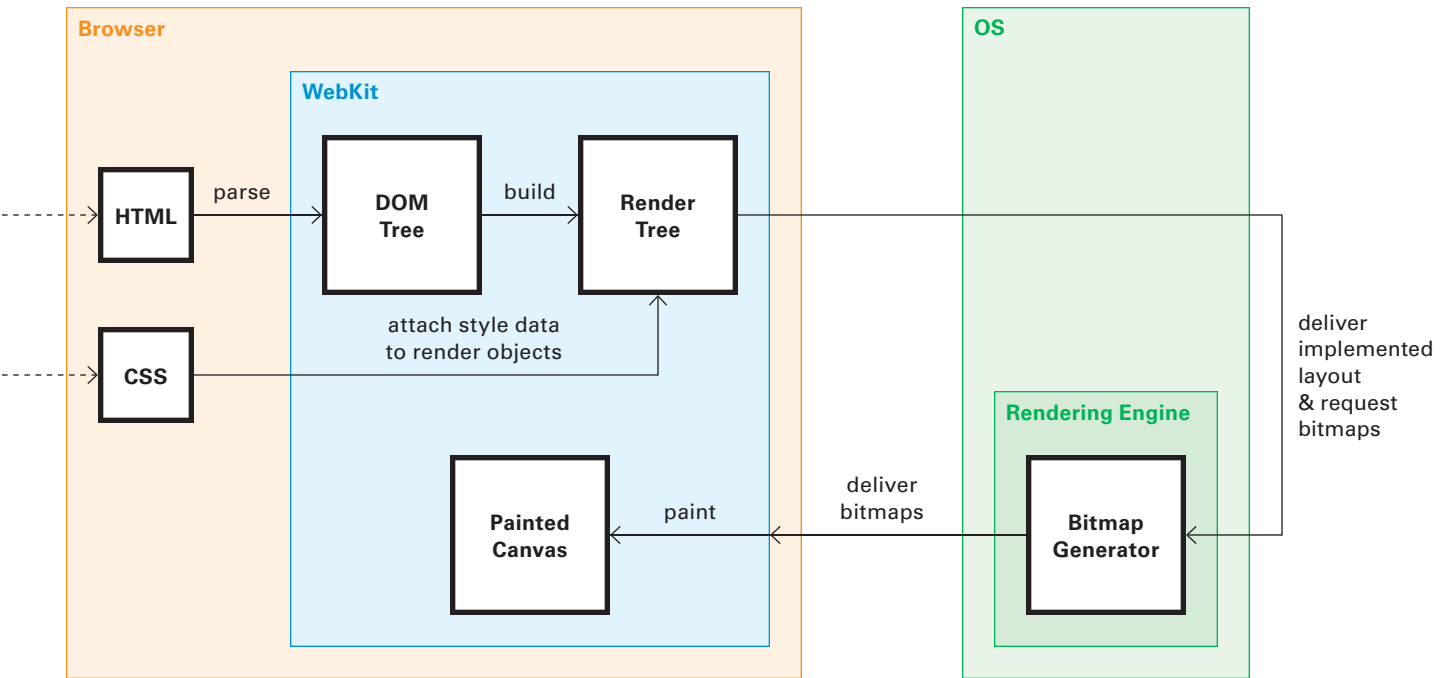
When a browser receives a document to be displayed, there are a number of steps it must go through to take the received content and turn it into what you see on screen. Most major browsers render a page by first parsing HTML into a DOM tree (see page 88) and then using that as the basis for a render tree (see page 89). The example shown to the right is based on WebKit and shows the two possible paths: one for simple text and the other for complex text.



The layout engine takes the DOM tree and attaches style data to it to create the render tree, which instructs the rendering engine on how to layout the content in the DOM tree.

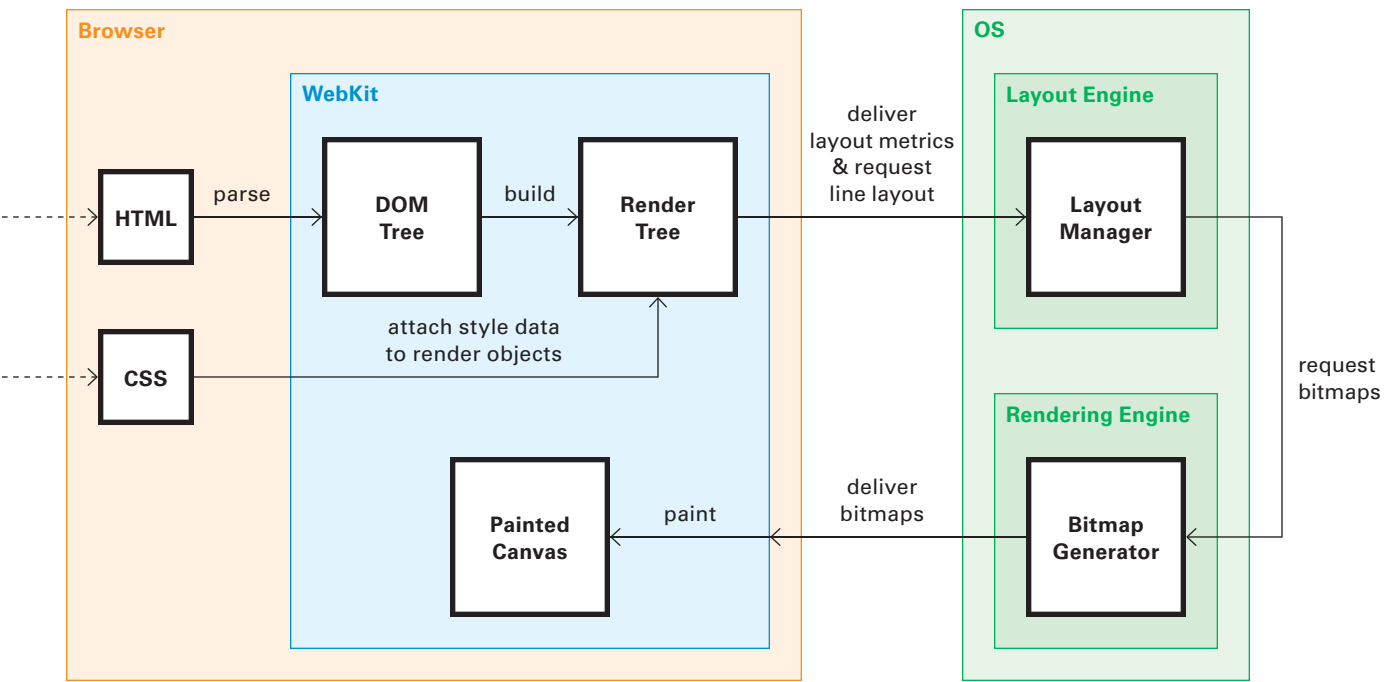
Path 1: Simple Text & Graphic Elements

Page rendering takes this path when dealing with non-text graphic elements and when the text to be laid out is “simple”, meaning it has no contextual shaping. Languages that have no contextual shaping in WebKit include English, Spanish, and Chinese. This means English text in WebKit *never* has ligatures.



Path 2: Complex Text

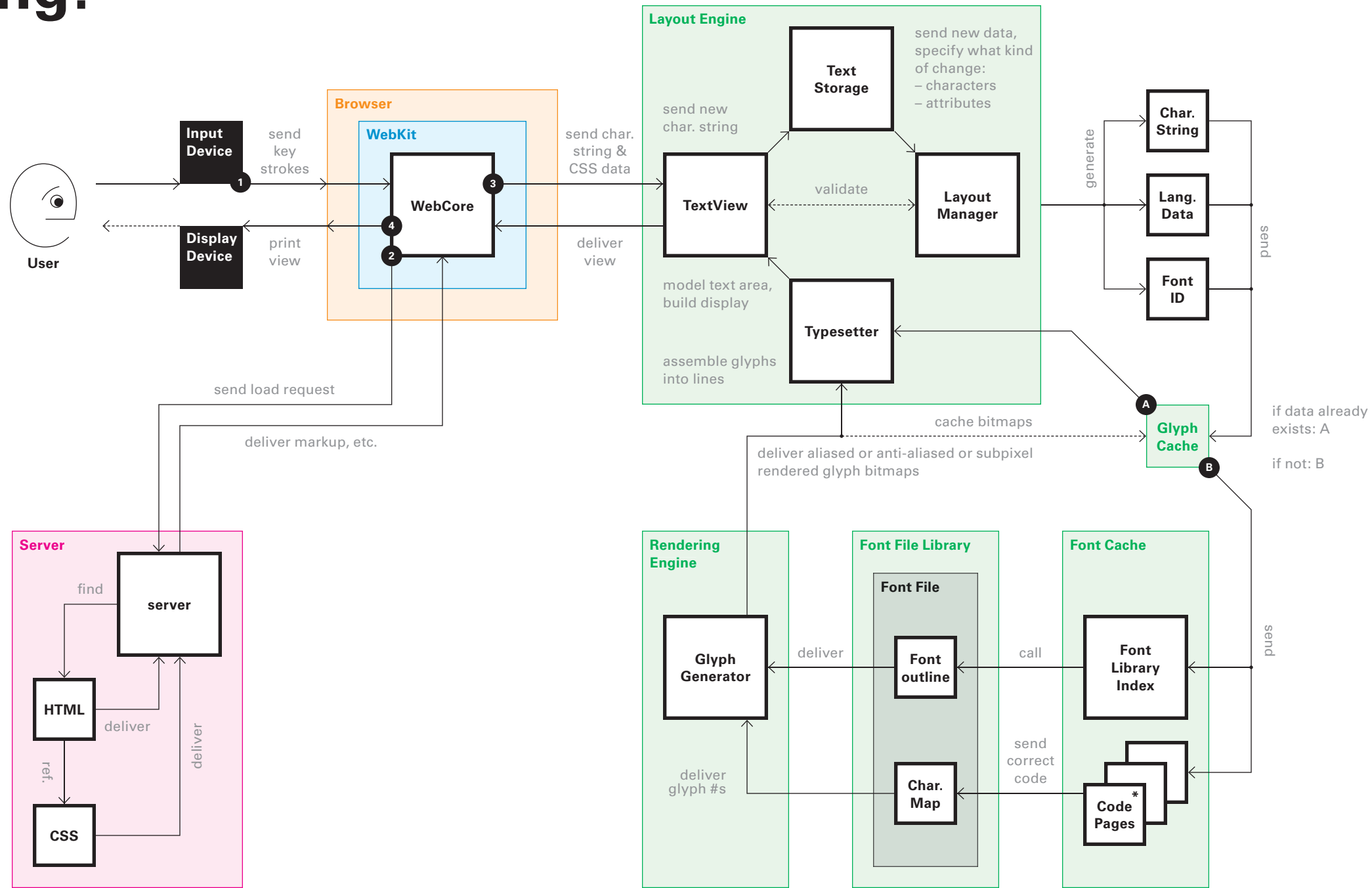
Page rendering takes this path when the text to be laid out is “complex”, meaning it has contextual shaping. Languages that have contextual shaping in WebKit include Arabic and all the Indic scripts. In this path, text line layout is handled by the operating system, not WebKit.



Webpage Rendering: Fonts

Font rendering in Web browsers is similar to font rendering in a text editor. (See page 96.) Both processes rely heavily on core OS APIs and resources to handle glyph rendering and, sometimes, text line assembling. The primary difference is that instead of the OS layout engine receiving text being input from the keyboard, it is receiving character strings pulled out of the markup that the server delivers to the Web browser. In addition, the browser engine also delivers style data to the rendering engine that the server provides in the form of both HTML markup tags and CSS attributes. This data is laid out by either the browser's internal layout engine or by the OS's layout engine depending on whether the text is simple or complex (see previous page). The OS rendering engine is always used to generate bitmaps.*

The example to the right, which is based on WebKit, follows path #2 from the previous page. If it was to follow path #1, all of the steps taken in the Layout Engine would be handled inside WebKit.



* It appears that Safari for Windows has its own rendering engine.

How Do Fonts Work on the Web?

Font Replacement

A number of techniques have been developed to get around the limited number of fonts available for the Web.

CSS image replacement (Pharck method)

This practice involves overlaying text with an image containing the same text written in the desired font. This is good for search engine optimization and aesthetic purposes but prevents text selection and increases bandwidth use.

Scalable Inman Flash Replacement (sIFR)

This is similar to image replacement techniques, though the text is selectable and rendered from vectors in Flash. However, this method requires the presence of a proprietary plugin on a client system. This method is an open source JavaScript and Adobe Flash dynamic webfonts implementation, enabling the replacement of text elements on HTML webpages with Flash equivalents. It was initially developed by Mike Davidson and improved by Mark Wubben. It is a scalable variety of HTML text-to-flash replacement pioneered by Shaun Inman.

Facelift Image Replacement (FLIR)

Similar to sIFR. But instead of using Flash, it embeds plain images that are generated automatically from the text on the webpage. Even if user does not have a Flash plugin installed, he will see the text replaced by FLIR. However, FLIR requires that the website host is capable of running PHP, and it is even less accessible than sIFR because the text is not selectable.

Cufón / Typeface.js

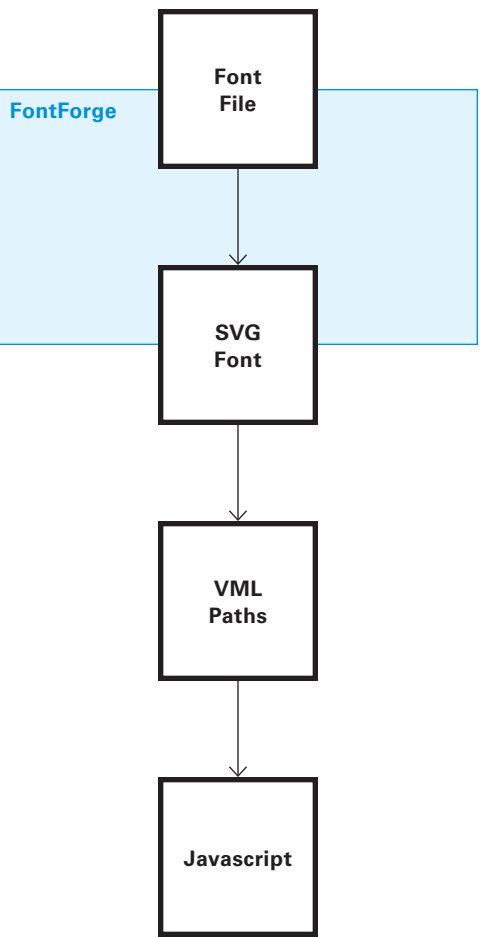
These are two similar but independent technologies using only JavaScript to draw the typeface onto the page using either the HTML5 <canvas> element for WebKit and Gecko browsers or VML for Internet Explorer

Source:
www.mightymeta.co.uk/introducing-the-web-safe-font-cheat-sheet/
www.mikeindustries.com/blog/sifr

The sIFR Process

- A normal HTML page is loaded into the browser.
- A JavaScript function is run which first checks that Flash is installed and then looks for designated tags, IDs, or classes.
- If Flash isn’t installed (or if JavaScriptis turned off), the HTML page displays as normal and nothing further occurs. If Flash is installed, JavaScript traverses through the page source measuring each element designated as something that can be “sIFRed”.
- Once measured, the script creates Flash movies of the same dimensions and overlays them on top of the original elements, pumping the original browser text in as a Flash variable.
- ActionScript inside of each Flash file then draws that text in your chosen typeface at a 6 point size and scales it up until it fits snugly inside the Flash movie.

The Cufón Generator Process



How Do Fonts Work on the Web?

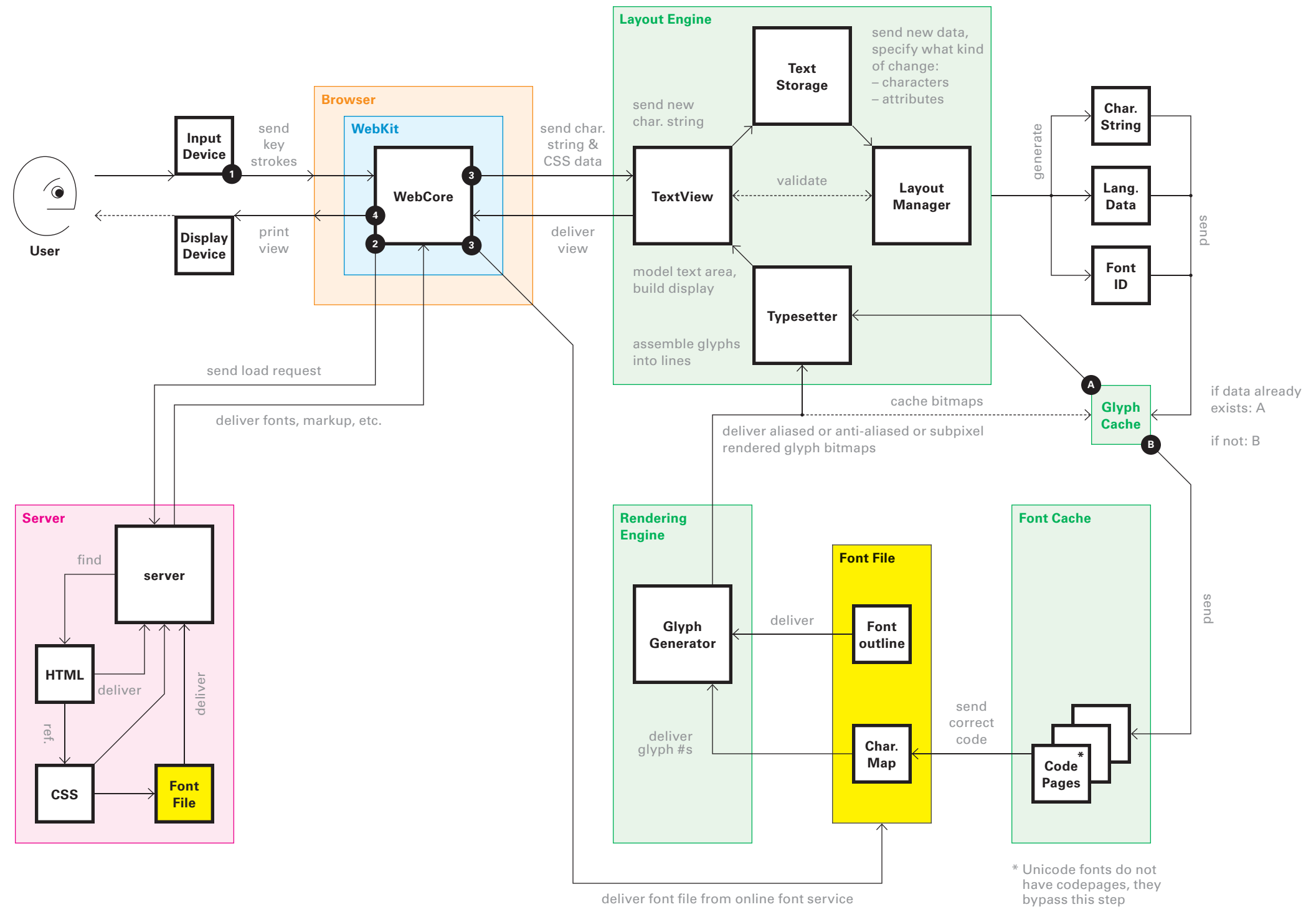
Font Hosting

A technique to download remote fonts was first specified in CSS2 through the use of the `@font-face` tag. It was quickly removed in CSS2.1 and not reintroduced until CSS3. It was (and remains) controversial because using a remote font as part of a webpage allows the font to be freely downloaded. This could result in fonts being used outside the terms of their license or illegally spread through the Web. TrueDoc (PFR), Embedded OpenType (EOT), and Web Open Font Format (WOFF) are formats designed to address these issues by creating fonts that do not work in any applications other than a Web browser.

In response to issues with `@font-face`, in the last two years a number of type foundries and independent companies have created online services for remote hosting and delivery of font files. Font hosting services allow users to pay either a subscription or a one-time purchase fee to host fonts online. This prevents unknown users from illegally downloading the source files. Most services host the font for the user and provide the necessary `@font-face` CSS declaration, although some use JavaScript instead.

`@font-face` code structure:

```
@font-face {  
  font-family: Gentium;  
  src: url(http://site/fonts/Gentium.ttf);  
}  
  
p {  
  font-family: Gentium, serif;  
}
```



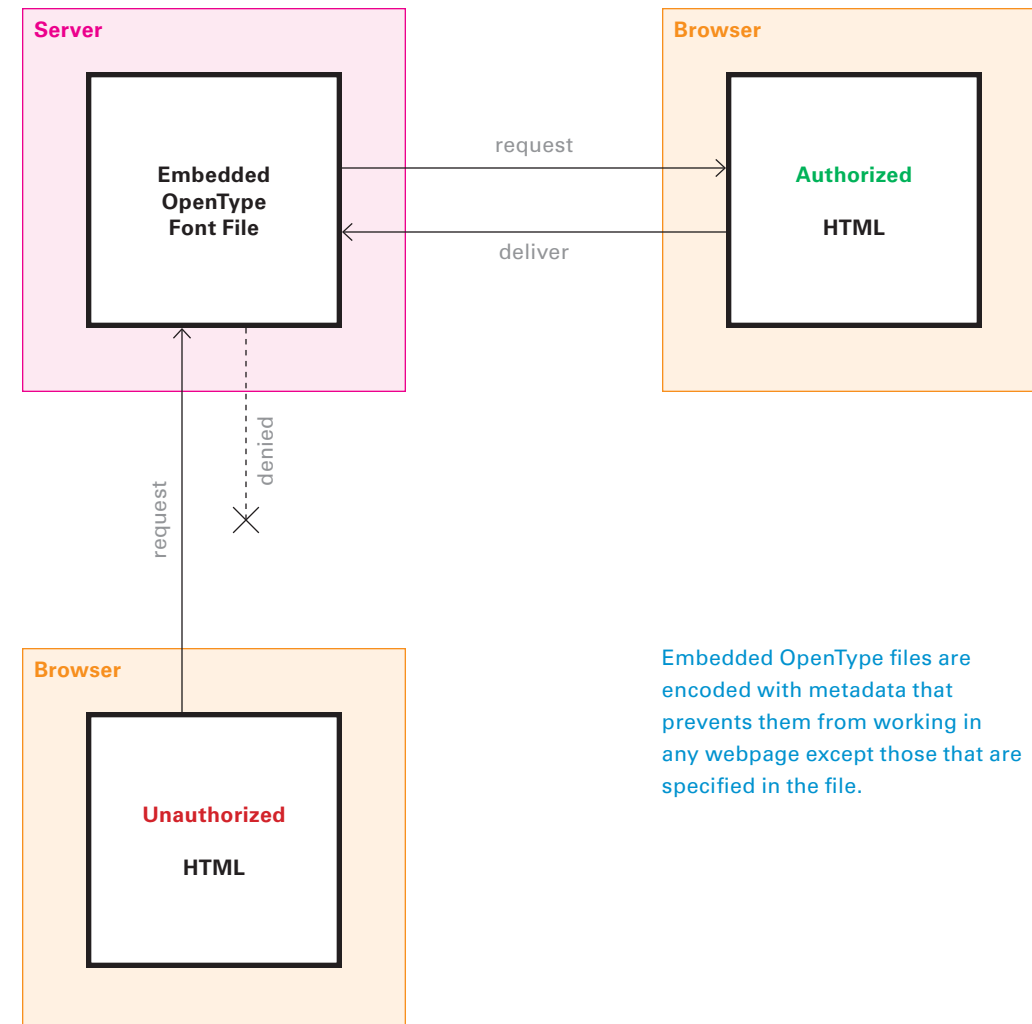
How Do Fonts Work on the Web?

EOT

Embedded OpenType (EOT) is a font file format developed by Microsoft to provide font embedding on the Web with font license protection. Font embedding on the Web is a term used only in conjunction with the EOT format; when any other font file format is used the process is called font hosting (see previous page).

There are two differences between standard OpenType (see pages 56 and 60) and Embedded OpenType. The first is that EOT encodes data into the font file that declares what website, Web directory, or webpage are allowed to access it. The second is that EOT uses compression and subsetting to decrease the font file size.

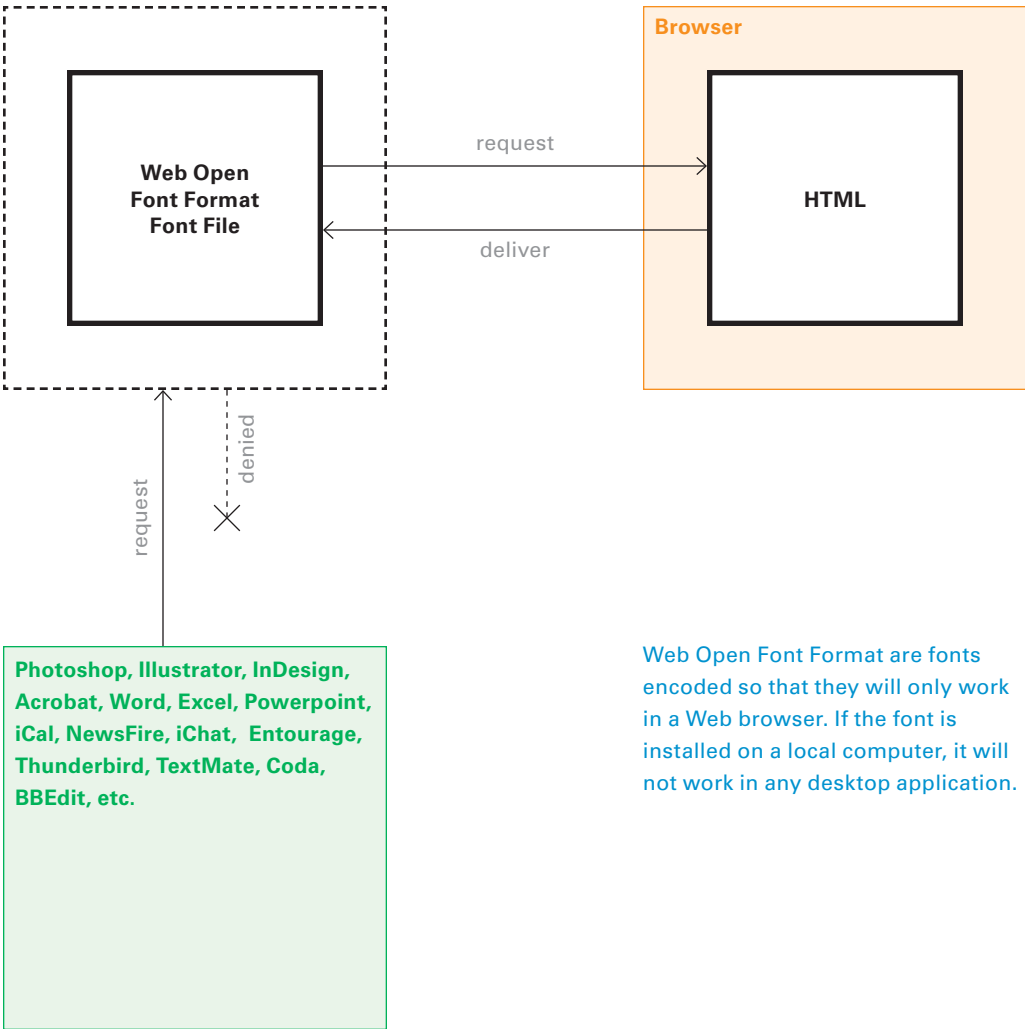
In 2008 Microsoft submitted EOT to the W3C in the hope of having it approved as a Web standard. In doing so they made the specification public and thus many critics argue that the encryption method used in EOT is now useless.



How Do Fonts Work on the Web?

WOFF

Web Open Font Format (WOFF), developed in 2009, is a wrapper for TrueType and OpenType fonts that allows them to be embedded in webpages. Unlike TrueType and OpenType fonts, which can be read by Web browsers and desktop applications, fonts using WOFF can only be read by Web browsers, thus providing a strong level of protection against piracy. In addition to the wrapper, the format is also compressed and can result in fonts that are up to 40% smaller than standard TrueType. The World Wide Web Consortium (W3C) has recommended that it become the *de facto* font format for use online.



Web Open Font Format are fonts encoded so that they will only work in a Web browser. If the font is installed on a local computer, it will not work in any desktop application.

Digital Rights Management (DRM) For Fonts

There are almost no technical barriers to hosting a font file and using it through @font-face. Since it would be easy for anyone with minor technical knowledge to download (i.e. steal the font file), most type foundry licenses don't allow use with @font-face. Seen in this light, font hosting services are essentially DRM services. There are many strategies to do this; however as of yet none of them are perfect. A determined, and technically savvy, user attempting to steal a font would eventually be able to do so. Current strategies used to protect fonts include:

- Webfont formats: Most of the services listed make use of Web formats for the typefaces they distribute (EOF, WOFF). This prevents the font from being used in desktop applications (Photoshop, InDesign, etc.). There are now several type foundries that sell their fonts in Web formats directly to users, allowing them to host these files as they see fit.
- HTTP referrer checking: Only authorized domains are allowed to link to fonts. Foundries which sell webfonts directly to users often require this measure be taken in the End User License Agreement (EULA) of the font.
- Obfuscation: Hard-to-guess file names listed as strings of seemingly random characters.
- Data URLs: Fonts are represented as Base64 encoded strings. Encoded fonts can be injected in-line with generated CSS. This makes font pirating difficult because there is no font file to steal; there is only raw data that would have to be re-compiled.
- Visible license data: Display of license, copyright, and “allowed” domains in the CSS. This serves as a reminder that repurposing font files is illegal.
- Segmenting and subsetting: Font files are split into multiple files and recombined using the CSS font stack. Additionally, only the characters needed for use on the page are delivered (preventing an unauthorized user from recombining segments into a complete font).

How Do Fonts Work on the Web?

SVG

Scalable vector graphics (SVG) is a graphics language based on XML (extensible markup language). Compared to the file size of bitmap image formats (.jpeg, .gif, etc.), **SVG is incredibly small**. Instead of storing color data for every pixel in the image file, vector graphics are **defined in terms of points, the lines/curves between them, and fills** (much like font file outline formats). This results in much less data needing to be stored in the file.

The SVG format online has many potential benefits, especially in the area of dynamic graphics and fonts because **unlike pixel-based data it can be scaled, rotated, or stretched without quality loss**. Since the image data is stored as vectors, scripting languages such as JavaScript can be used to manipulate the image data (e.g. page location data can be treated as a variable instead of a fixed value) in real-time.

Bitmap

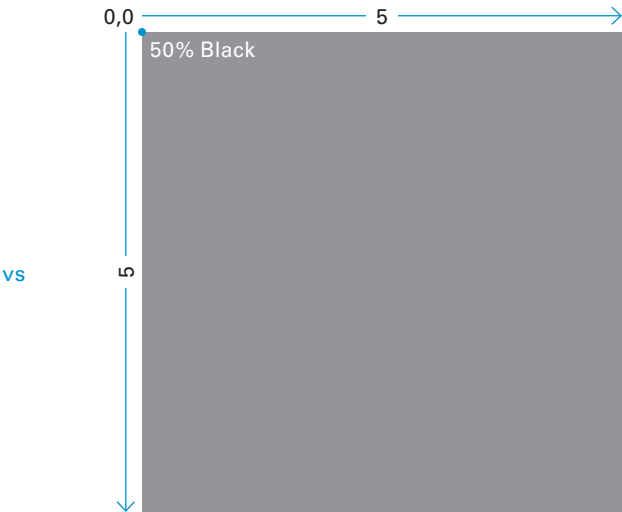
25 pixels × (location + color) = **50 datapoints**

0,0 50% Black	1,0 50% Black	2,0 50% Black	3,0 50% Black	4,0 50% Black
0,1 50% Black	1,1 50% Black	2,1 50% Black	3,1 50% Black	4,1 50% Black
0,2 50% Black	1,2 50% Black	2,2 50% Black	3,2 50% Black	4,2 50% Black
0,3 50% Black	1,3 50% Black	2,3 50% Black	3,3 50% Black	4,3 50% Black
0,4 50% Black	1,4 50% Black	2,4 50% Black	3,4 50% Black	4,4 50% Black

Bitmap image formats contain data specifying color for every pixel location – with a large block of the same color, there is a large amount of redundant data because there are two pieces of data for every pixel.

SVG

(origin) + (x width) + (y height) + (color) = **4 datapoints**



SVG format allows for the same shape to be defined using only four datapoints!



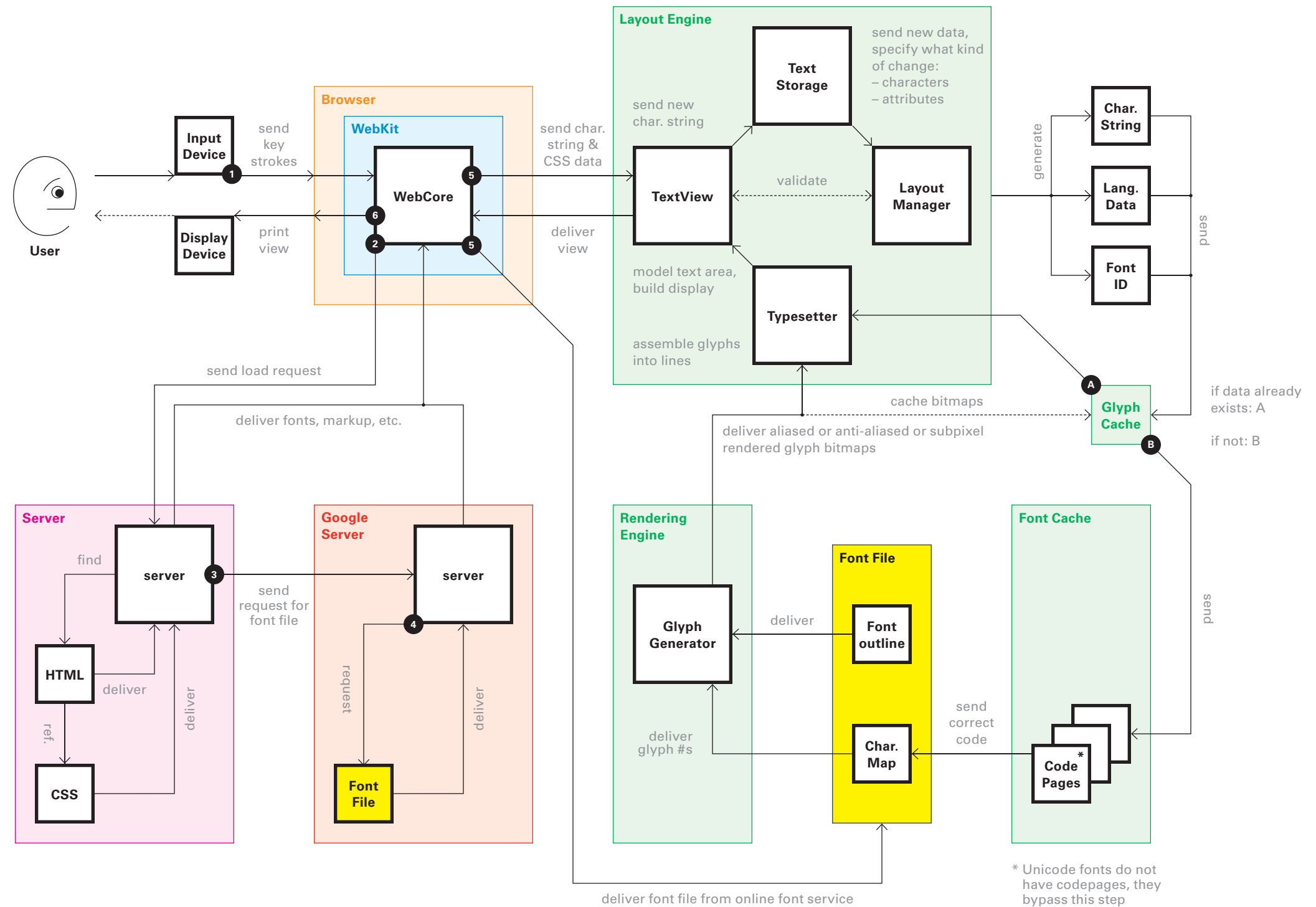
Scripting languages such as JavaScript can be used to dynamically manipulate the image data of an SVG file. Location, size, or color data can be treated as variables rather than fixed entities.

How Do Fonts Work on the Web?

Font Hosting: Google

The Google Font Directory is a pool of free webfonts available under the OFL or Apache License. The fonts are hosted by Google and can be linked within a webpage using a simple CSS call like:

```
<html>
<head>
  <link rel="stylesheet" type="text/css"
  href="http://fonts.googleapis.com/
  css?family=Tangerine">
  <style>
    body {
      font-family: 'Tangerine', serif;
      font-size: 48px;
    }
  </style>
</head>
<body>
  <h1>Making the Web Beautiful!</h1>
</body>
</html>
```



How Do Fonts Work on the Web?

Font Hosting: Typekit

Typekit appears to be the most popular font hosting service at the moment. They charge a yearly subscription fee that opens the entire library to publishers, who are then able to select what typefaces they want to use on what sites (both must be specified). **Typekit does more than host files for @font-face use, the service also implements a number of strategies for making font hosting safer** such as subsetting and fragmenting font files, validating that a given site is permitted to use a font, and using webfont formats. Additionally, they link to the font file through JavaScript instead of plain @font-face.

