Electronic Document Preparation Pocket Primer

Vít Novotný

December 4, 2018

Contents

Introduction

With the advent of the digital age, typesetting has become available to virtually anyone equipped with a personal computer. Beautiful text documents can now be crafted using free and consumer-grade software, which often obviates the need for the involvement of a professional designer and typesetter. The level playing field of the Internet coupled with the rising popularity of digital-only documents then allows the author to bypass the publisher as well, if they so wish, without jeopardizing their chance of recognition.

The aim of this book is to provide a general overview of the tools and techniques tied with writing, designing, typesetting, and distributing text documents—one of the principal means of knowledge preservation and transfer known to man. Each chapter describes one discrete step of document preparation along with practical examples and references to literature for those interested in further study.

The chapter are filled with examples that illustrate the subject matter. These should be consulted whenever the concepts described in the text are unclear to the reader. Although care was taken not to favor any computing environment, some examples feature utilities for Unix and Unix-like operating systems. These utilities may or may not have a suitable counterpart in operating systems such as Windows; To try the corresponding examples out, the reader is advised to install a free Unix-like environment—such as Cygwin for Windows—on their computer.

Chapter 1

Writing

The essence of a document is the idea it represents. In the case of a text document, this idea is articulated through speech, which is transcribed using text, optionally accompanied by figures, and then laid out on a sheet of paper according to a design. Since the text is typically independent on the design, whose task is to support and elicit the internal structure of the text, it is writing that is the logical first step in the text document creation.

The essentials of writing in any given natural language include *grammar rules*, which specify the structure of spoken language, and *orthographic rules*, which impose additional requirements on written text. The complexity of either set of rules depends entirely on the language in question. Some writing systems, such as those that incorporate Chinese characters, are not phonographic and the correspondence between the spoken words and the written symbols needs to be memorized by the writer on a word-to-word basis. Other languages may use vastly different grammar rules for speaking and for writing, which means that a spoken sentence needs to be translated first before writing down. A writer needs to recognize these specifics.

On top of grammar and orthographic rules stand *style guides*, which, in order to improve consistency, codify how common language patterns are encoded. More comprehensive style guides—such as *the Chicago Manual of Style* or *the Oxford Style Manual*—often go beyond writing and provide guidelines on design and type-

This document was prepared in accordance with William Strunk's *Elements of Style*, an American English style guide for general use.

Zwei Trichter wandeln durch die Nacht.
Durch ihres Rumpfs verengten Schacht
fließt weißes Mondlicht
still und heiter
auf ihren
Waldweg
u.s.
w.

Figure 1.1: Exceptions that prove the rule about the separation of text and design can sometimes be encountered in poetry. Above is Christian Morgenstern's *Trichter*, where the text and its form are intimately intertwined.

setting as well, making them an indispensable reference on the editorial tradition.

Above all stand the *typographic rules*, which specify how the resulting document should be typeset so that it doesn't disturb the eye of the reader. These, as well as the orthographic rules on hyphenation, can be left out of consideration during writing, as it is the page that should be formed around the writing and not the other way around.

1.1 Text Processing

Originally the domain of the pen, the quill, the stylus, and the more recent typewriter machine, manuscripts of today are produced mainly using the personal computer and stored in *text files*. The discipline of creating and manipulating digital text is called *text processing* and will be the focus of this section.

1.1.1 Character Encoding

Although computing at its most primal has no use for anything but numbers, it has nevertheless been accompanied by text from the very outset. Even the earliest computers from 1950s were programmed with both raw machine code and the text programming language of *the FORmula TRANslator* (FORTRAN). The digital representation of letters, digits and other characters was initially closely

tied to each specific application and processor architecture, but with the advent of computer networking in 1960s, mutual intelligibility became a point of concern. "We had over sixty different ways to represent characters in computers. It was a real Tower of Babel," explains Bob Berner [1], an American computer scientist who worked at IBM during 1956–1962 and who drafted *the American Standard Code for Information Interchange* (ASCII) [2]—a *character encoding* from 1963 that unified the digital representation of text across the computer industry and enabled computer networking on a large scale.

EBCDIC by IBM was the default encoding on IBM's System/360 mainframes and was in active use until the introduction of PC in 1981. In writing systems using Chinese characters, special encodings, such as Big5, JIS, and EUC, are used to this day. For brevity, the text focuses on the main stream of international encodings.

ASCII

In ASCII, every character is represented by a number from zero to 127, which is transformed to a seven-bit integer called a *character code*. These 128 codes are used to encode *printable characters*—spanning the letters of the English alphabet, digits, punctuation, and other symbols—and *control codes*, as depicted in Table 1.1. Unlike printable characters, control codes have no fixed visual representation and they were used to implement application-specific communication protocols and text formatting; their precise semantics were defined in a much later standard from 1972 [3]. Unconstrained by the bandwidth and the storage limitations of the 1960s and 1970s, today's communication protocols and text formats gravitate towards markup constructed from printable characters, which, unlike control codes, are easy to read and write by humans.

The following properties make it easy to manipulate and reason about character strings encoded in ASCII:

- Each character is represented by exactly seven bits. This makes it easy to allocate space for character strings of fixed length, to measure the number of characters stored in a memory region, and to perform basic operations, such as adjacent character retrieval or text truncation.
- Characters are alphabetically ordered. Character strings can therefore be collated by comparing character code binary values.
- Lowercase and uppercase letters, digits and control codes form contiguous ranges of character codes. This simplifies classification.

7				0	0	0	0	1	1	1	1
6	*Bits*			0	0	1	1	0	0	1	1
5				0	1	0	1	0	1	0	1
4	3	2	1	*Ctrl codes*		*Symbols*		*Upper case*		*Lower case*	
0	0	0	0	NUL	DLE	␣	0	@	P	‘	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	”	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	S
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	’	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	q	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

Table 1.1: The ASCII encoding, as specified in the 1986 revision of the standard. [4]

Code point range	*Encoding*
0–127	0␣␣␣␣␣␣␣
128–2047	110␣␣␣␣␣ 10␣␣␣␣␣␣
2048–65,535	1110␣␣␣␣ 10␣␣␣␣␣␣ 10␣␣␣␣␣␣
65,536–1,114,111	11110␣␣␣ 10␣␣␣␣␣␣ 10␣␣␣␣␣␣ 10␣␣␣␣␣␣

Table 1.2: The UTF-8 encoding. Each ␣ represents one bit of the UCS code point in binary.

Character	*Code point encoding*				
Ř	344	⇝	101011000	⇝	11000101 10011000
e	101	⇝	1100101	⇝	01100101
č	269	⇝	100101000	⇝	11000100 10101000

Table 1.3: An example of the UTF-8 encoding

- There is precisely one way to encode any printable character. The conversion between the lower- and uppercase letters is a matter of inverting one bit.

  This comes at the expense of support for non-English writing systems. As a temporary workaround, a set of ASCII derivatives that replaced the less-needed characters of # $ @ [\ ] ^ ‘ { | } and ~ for international characters was specified in the ISO 646 standard from 1972. [3]

Eight-bit Encodings

With the byte size stabilizing at eight bits, new character encodings emerged that were based on ASCII and used the additional bit to encode characters of non-English writing systems while retaining complete backwards compatibility with ASCII. Beside the numerous vendor-specific encodings (called *code pages*), a set of fifteen eight-bit encodings covering all major modern writing systems whose characters fit within the space of 128 additional combinations was standardized in the ISO/IEC 8859 series released during 1986–2001.

Compared to ASCII, eight-bit encodings introduced an additional level of complexity to text processing:

- Each character is exactly eight bits wide. The manipulation with strings is therefore as straightforward as with ASCII.
- Character strings can no longer be collated by character code comparison. Each encoding requires separate collation tables.
- Classes of characters, such as uppercase and lowercase letters or punctuation, no longer form contiguous ranges and their position varies among encodings. This impedes character classification.
- Idiosyncrasies, such as the ligature of æ and invisible hyphenation hints, are included in several encodings, which makes it more difficult to determine character string equivalence. Algorithms for case conversion vary among encodings.
- There exists no standard mechanism to detect which encoding is being used. The distinction needs to be done on the application level using either heuristics, additional metadata, or human intervention. Consequently, no standard mechanism exists to use different character encodings within a single text document.

A portion of this complexity is inherent in the task of encoding the characters of all modern writing systems, but the overhead caused by the character encoding fragmentation proved to be unnecessary.

The Universal Character Set and Unicode

In the early 1990s, the continual increase in the available bandwidth and storage led to the creation of the standards of Unicode [5, 6] and *the Universal multiple-octet coded Character Set* (UCS) [7] in an attempt to create a text encoding that would contain the characters of all the world's languages and succeed ASCII as the *lingua franca* of text interchange.

UCS is an ever-expanding catalogue of characters from writing systems both modern and ancient, and symbols ranging from diacritical marks, punctuation, and ideograms to mahjong tiles, alchemical symbols, and the ancient Greek musical notation. Each of these characters is assigned a number, called a *code point*, ranging from 0 to 2,147,483,647 (`7F FF FF FF` in the hexadecimal notation) with the numbers of the most common characters in the range from 0 to 65,535 (`FF FF`) called *the Basic Multilingual Plane* (BMP). The smallest unit of division in UCS are *blocks*, which contain 256 thematically related characters. UCS encodings map code points to binary character codes and vise versa.

Notable are also the seven-bit encodings of UTF-7 and Punycode, which bring Unicode support to protocols that were designed with the seven-bit ASCII in mind, such as e-mail.

Three major encodings are specified in the UCS standard and its amendments [8, 9]:

1. UTF-32 directly encodes UCS characters by transforming their code points to four-byte integers. UTF-32 is also known as UCS-4.
2. UTF-16 directly encodes characters within BMP by transforming their code points to two-byte integers. Code points in the range from 65,536 to 1,114,111 (`01 00 00`–`10 FF FF`) are transformed into pairs of two-byte integers, called *surrogate pairs*, ranging from 55,296 to 57,343 (`DC 00`–`DF FF`). To enable the UTF-16 encoding, the code points in this range will never be assigned to characters [10, sec. 3.4, D15]. The same is true of code points above 1,114,111 (`10 FF FF`), which allows UTF-16 to encode any UCS character.
3. UTF-8 directly transforms code points ranging from 0 to 127 (`7F`) to one-byte integers. Since the first UCS block of the BMP matches ASCII, any text encoded in eight-bit ASCII is also encoded in UTF-8. Code points in the range from 127 to 1,114,111 (`00 00 7F`–`10 FF FF`)

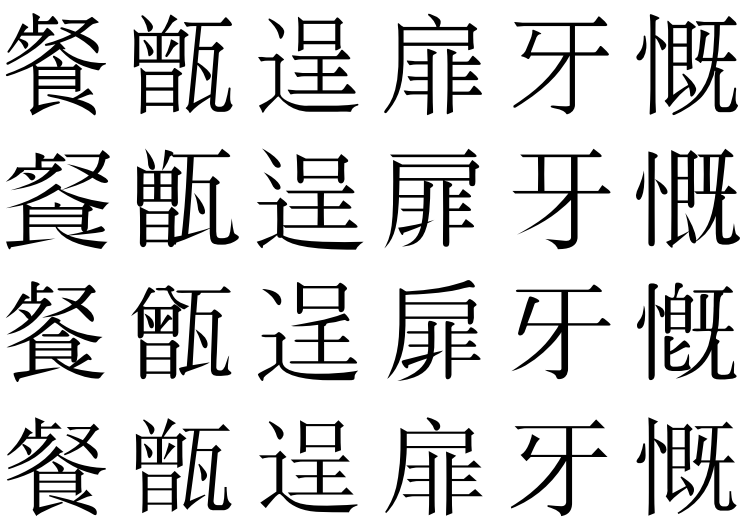


Figure 1.2: Several Han characters in the traditional Chinese, Japanese, Korean, and Vietnamese variants

are transformed into two to four one-byte integers ranging from 128 to 253 (80–FD). The encoding is illustrated in tables 1.2 and 1.3.

UTF-32 is primarily used for the fixed-space internal representation of individual UCS characters inside programs, UTF-16 fulfills a similar role in programs that only work with BMP, and UTF-8 is used for text storage and interchange. Since 2010, the majority of text content on the Web has been encoded in ASCII and UTF-8. [11]

Unicode was a competing standard for universal text encoding that underwent a merger with UCS in version 1.1 and since then, the standards have been kept closely synchronised. Unicode is a superset of UCS which defines additional information about UCS characters—such as their general category, directionality, case, or numeric value [10, sec. 3.5 and ch. 4]—, various text processing algorithms, and implementation guidelines.

Regarding text processing, Unicode and UCS represent a compromise between the simplicity of the seven-bit ASCII and the heterogeneity of eight-bit encodings:

One of the design goals of UCS was to avoid assigning code points to different glyphs that carry the same meaning. As a result, the visually distinctive Han characters used in the East Asian countries of China, Japan, Korea, and Vietnam were merged into a set of 75,960 ideograms in a process referred to as the *Han Unification* [10, sec. 18.1]. This simplifies text processing, but also makes it impossible to encode a text in multiple East Asian languages without having to rely on external markup to select appropriate regional fonts. As a result, a derivative of UCS that doesn't implement the Han Unification was developed for use in operating systems based on *the Real-time Operating system Nucleus* (TRON) and is used in the East Asia alongside UCS and region-specific encodings.

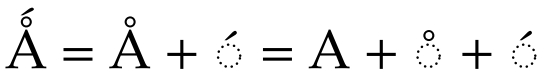


Figure 1.3: Some UCS characters can be either input as a single entity or composed from several combining characters. Regarding Unicode normalization forms, all of the above representations are canonically equivalent.

```
iconv -f latin2 -t utf8 -- old.txt > new.txt
```

Figure 1.4: Text files can be converted between encodings using the `iconv` command-line tool. The sample code shows the file `old.txt` being converted from the ISO/IEC 8859-2 encoding to UTF-8. The result of the conversion is stored in the file `new.txt`.

- If simple text manipulation is preferred over space efficiency, each character can be made exactly two or four bytes wide using the UTF-16 and UTF-32 encodings.
- Although character strings can not be collated by a simple character code comparison, a collation algorithm is defined in the Unicode specification [12] and collation tables for major locales [13] are maintained by the Unicode Consortium.
- Classes of characters—such as uppercase letters, lowercase letters, numbers, and punctuation—do not form contiguous ranges, but their position is directly specified in the standard [10, sec. 4.5].
- Although idiosyncrasies—such as ligatures, invisible hyphenation hints, and combining characters—are present in UCS, explicit normalization algorithms for character string equivalence testing are specified by the standard [10, sec. 2.12]. An algorithm for case conversion is also specified [10, sec. 3.13].
- The BYTE ORDER MARK (FE FF) character can be inserted at the beginning of a text as a signature of Unicode encodings. As the name suggests, the order in which the FE and FF bytes arrive also indicates the order of bytes (called *endianity*) that was used to encode integers. In UTF-32 and UTF-16, endianity can be chosen arbitrarily by the encoding application. In UTF-8, one-byte integers are used and the notion of endianity is therefore meaningless.

Figure 1.5: Text input methods are not limited to keyboard layouts. Software that enables the input of non-Latin characters on a keyboard through reversed romanization can often be the best option for writing systems with a large number of characters. Above is the Google Pinyin input method for the Android operating system, which makes it possible to input Chinese characters using the pinyin phonetic system.

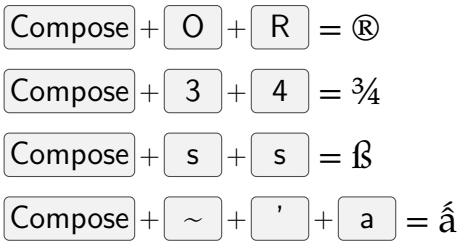


Figure 1.6: The *Compose* key followed by a mnemonic sequence of ASCII characters produces a UCS character. Although originally a physical key, *Compose* is not available on modern PC and Apple keyboards and is usually mapped to the right *Ctrl* or *Super* key in software. *Compose* is natively supported on Unix and Unix-like operating systems using the X Window System. On other operating systems, support can be added by third-party software.

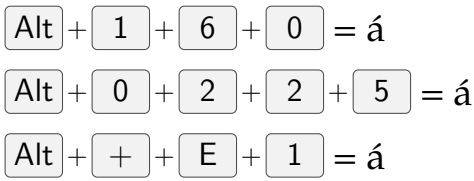


Figure 1.7: On the Windows operating system, holding the *Alt* key and typing a sequence of numbers produces a character with the corresponding number from either an IBM code page, if the number has no leading zero, or from a Windows code page otherwise. The code pages vary depending on the current locale; in English locales, the IBM code page 437 and the Windows code page 1252 are used. After a Windows Registry modification, it is also possible to directly produce UCS characters by holding the *Alt* key and typing the corresponding UCS code point in hexadecimal.

1.1.2 Text Input

To insert text into a document, it is necessary to use an input device. In case of personal computers, this is typically a computer keyboard and a mouse, although the ongoing research in the areas of *Sound Recognition* (SR) and *Optical Character Recognition* (OCR) makes it possible to use a microphone or a tablet as well. On handheld devices, the use of either a numeric keypad or a touch-screen is more typical.

An operating system will typically provide one or more input methods for each input device through a component commonly referred to as the *Input Method Editor* (IME). The ASCII encoding was developed with typewriters and teleprinters in mind and, as their direct descendant, the standard computer keyboard provides support for all ASCII characters. This doesn't apply to the much larger UCS and it is the task of an IME to provide a mechanism for the creation and selection of keyboard layouts that will allow the user to input any UCS character. Some programs may provide input methods of their own that are independent on the IME.

1.1.3 Text Editors

A *text editor* is an application that can be used to create and modify text files. Entry-level text editors are often distributed with an operating system and offer little beyond the ability to load, modify, and save text files in a text encoding of choice. Entry-level text editors with a *Graphical User Interface* (GUI) include the free Leafpad for GNU/Linux and *the Berkeley Software Distribution* (BSD) family of operating systems, and the proprietary Notepad for Windows and TextEdit for Mac OS. Entry-level text editors with a *Command Line Interface* (CLI) include the free `joe`, GNU `nano`, and `pico`.

More advanced text editors come with the support for *regular expressions* and *version control*—which will be covered in sections 1.1.5 and 1.2—and user modules that extend the base functionality. Advanced GUI text editors include the free Notepad++ and Atom, and the proprietary Sublime Text. Advanced CLI text editors include the free Emacs, `vi`, and `vim`. These CLI text editors are notorious for their steep learning curve; in exchange, they empower the users to perform complex text editing.

1.1.4 Interactive Document Preparation Systems

Interactive *Document Preparation Systems* (DPSes) are a breed of text editors that produces fully-formatted text documents instead of (or along with) text files. The reader is advised to avoid interactive DPSes that use proprietary, undocumented, or obscure file formats which lock the user into using the respective DPS. Well-defined interactive DPS file formats include *the Portable Document Format* (PDF) [14], *the Office Open XML format* (OOXML) [15], and *the Open Document Format for office applications* (ODF) [16].

The primary difference between text editors and DPSes is the fact that the user is expected to use the DPS to mark up, design, and typeset the resulting text document, whereas with plain text files a multitude of choices is available at each step of the document preparation process. The self-sufficient nature of DPSes may be a time-saving feature for simpler documents, but in the case of more complex documents, the markup and typesetting capabilities of a DPS may not be up to par with those of a dedicated tool. Interactive DPSes include the free Apache OpenOffice and Scribus, and the

proprietary TextEdit, Microsoft Word, Scribus, Adobe InDesign, Adobe FrameMaker, and QuarkXPress.

1.1.5 Regular Expressions

The *Chomsky hierarchy* is a classification of text production rule sets (called *formal grammars*), which was proposed [17] in 1956 by the American linguist Noam Chomsky in his endeavor to discover a good formal model for the description of natural languages. The class of *regular grammars*, which is the least powerful of the proposed classes, and the related formal model of *regular expressions* enable the writer to match patterns within text.

Since regular expressions are just a formal model, a software implementation needs to settle on a concrete syntax. One of the earliest standard syntaxes are *the Basic Regular Expressions* (BRE) and *the Extended Regular Expressions* (ERE) syntaxes [18, part 1, ch. 9] described in Table 1.4, which are supported by most text processing programs on Unix and Unix-like operating systems.

More extensive syntaxes include the GNU extensions of BRE and ERE, the regex syntax of the Perl programming language, and their derivatives. For these syntaxes, the term *regular* is a misnomer, as they can be used to describe formal grammars that, according to the Chomsky hierarchy, are stronger than regular. To disambiguate the term, expressions in these syntaxes are often called *regexes*.

Mastering Regular Expressions [19] by Jeffrey E. F. Friedl is an extensive resource on regexes.

Many regex syntaxes and the software that implements them were designed for the processing of ASCII text and may behave in surprising ways, when confronted with UCS characters. The software may assume that each character is exactly one byte wide and fail to recognize any character that occupies several bytes. It may also assume that all UCS characters fall within BMP and exhibit the same problem with characters outside BMP. More subtle, but no less precarious, can be the lack of support for Unicode case conversion and normalization algorithms, which makes it difficult to perform robust case-insensitive matching and the matching of characters that can be encoded in several different ways. The lack of awareness of the invisible characters that can appear in UCS text—such as the ZERO WIDTH SPACE (20 0B), ZERO WIDTH NON-JOINER (20 0C), ZERO WIDTH JOINER (20 0D), and ZERO WIDTH NO-BREAK SPACE (FE FF)—, is also problematic and can lead to false negative matches. Conversely, modern regex syntaxes that at

BRE *regex*	*Description*	*Matches*
`we\{1,2\}p`	The repetition expression in the form of $c\{m,n\}$ matches the character c repeated $k \in \langle m; n\rangle$ times. Other forms include $c\{m,\}$ for $k \in \langle m; \infty\rangle$ and $c\{m\}$ for $k = m$.	weeps, wept
`e*ne`	Star (`*`) is a *repetition operator* equivalent to the interval expression of `\{0,\}`.	never, enemy, Kleene
`\(`⟨*regex*⟩`\)`	A *subexpression* is a parenthesized regex. Any interval expression or repetition operator used immediately after a subexpression applies to the entire parenthesized regex.	⟨*regex*⟩
`^ar`	At the beginning of a regex or a subexpression, a caret (`^`) matches the beginning of a string.	argument, arrow keys
`ore$`	At the end of a regex or a subexpression, the dollar sign (`$`) matches the end of a string.	iron ore, dumbledore
`be.`	A period (`.`) matches any single character.	or not to be?
`be[ea]`	A *matching list expression* is enclosed in square brackets (`[ ]`) and contains a list of characters that the bracket expression matches. It may contain other entities omitted here for brevity.	beehive, grizzly bear, glass beads
`be[^ea]`	A *non-matching list expression* contains a caret (`^`) as its first character and matches any character that the corresponding matching list expression would not match.	obeah, bend, libela
`\^\*\.\\\$`	Backslash (`\`) is an *escape character* that either suppresses or activates the special meaning of the following character.	^*.\$
`\(..\).*\1`	A *backreference* in the form of an escaped number $n \in \langle 1; 9\rangle$ (`\1`, `\2`, …, `\9`) matches anything the nth subexpression matched.	ara arara, dardanelles, nationality

Table 1.4: An informal description of the BRE syntax (above) and the differences in the ERE syntax (below)

ERE *regex*	*Description*	*Matches*
`we{1,2}p`	Unlike in BREs, braces aren't escaped.	weeps, wept
`pe+rl?`	The plus sign (`+`) and the question mark (`?`) are repetition operators equivalent to the interval expressions of `\{1,\}` and `\{0,1\}`.	persona, peer, speech, perl
`(`⟨*regex*⟩`)`	Unlike in BREs, parentheses aren't escaped.	⟨*regex*⟩
`(on|t).`	Vertical line (`|`) is an *alternation operator* that separates multiple regexes. The whole regex matches any of the alternative regexes.	one, two, trophy, truth
`(..).*\1`	EREs do not support backreferences.	⟨*undefined*⟩

Regex	*Description*
`\x{`⟨*n*⟩`}`	Matches the UCS character with code point ⟨*n*⟩ in hexadecimal.
`\N{`⟨*n*⟩`}`	Matches the UCS character, whose `Name` property, `Name_Alias` property, or code point label tag equals ⟨*n*⟩.
`\p{`⟨*p*⟩`}`	Matches any UCS character with property ⟨*p*⟩.
`\P{`⟨*p*⟩`}`	Matches any UCS character without property ⟨*p*⟩.
Property	*Description*
`Letter`	This property is satisfied by any letter.
`Punctuation`	This property is satisfied by any punctuation.
`Symbol`	This property is satisfied by any symbol.
`Mark`	This property is satisfied by any mark.
`Number`	This property is satisfied by any number.
`Separator`	This property is satisfied by any separator.
`Other`	This property is satisfied by any UCS character that doesn't belong to any of the abovelisted categories.
`Block=`⟨*b*⟩	This property is satisfied by characters that reside in the UCS block ⟨*b*⟩. UCS blocks include `Basic Latin`, `Greek`, `Arabic`, etc.
`Script=`⟨*s*⟩	This property is satisfied by characters that belong to the writing system ⟨*s*⟩. Writing systems include `Latin`, `Korean`, `Chinese`, etc.
`Numeric Value=`⟨*n*⟩	This property is satisfied by any UCS character with the numeric value ⟨*n*⟩.

Table 1.5: The elements of the Unicode regex syntax implemented by Perl 5.2 and Java 7. The list of properties is not exhaustive.

least partially implement the Unicode standard for regular expressions [20]—such as those of Perl 5.2 or Java 7—are actively aware of UCS and provide features that enable the matching of characters based on their general category, numeric value, directionality, and other properties defined by Unicode, as shown in Table 1.5.

The authoritative resource on `grep`, `sed`, and `awk` is *Sed & awk* [21], which explains each program as well as the BRE and ERE syntaxes in full detail.

The most elementary text processing CLI program is `grep`, which makes it possible to search text files for fixed strings and regexes in default of an advanced text editor. Unless configured otherwise, the tool will present lines that contain one or more matches to the user. A more advanced text-processing CLI program is `sed`, which features a simple programming language that can be used to arbitrarily search and transform text files. `Awk` is a CLI program that also features a text-processing programming

language, albeit a more advanced one than that of `sed`. Originally developed for the Research Unix during 1973–1977, `grep`, `sed`, and `awk` are available in various flavors for most operating systems.

1.2 Version Control

When writing a text document, it is often useful to have a backup of the previous versions of files, so that undesirable changes can be reverted whenever necessary. If more than one person contributes to the document, the ability to track the authorship of these changes also becomes an asset. At their most rudimentary, *Version Control Systems* (VCS) record changes along with their descriptions and authorship information. These changes can then be viewed, and reverted. With a single contributor, VCS are a convenient alternative to manual version archival. With several contributors, VCS become an essential tool.

VCS can be dichotomized based on their architecture, which is either *centralized* or *decentralized* . Centralized VCS store all versions in a repository located on a remote server. Users send new versions to the server and retrieve existing versions using a client software. The client software is *thin* in the sense that it does not store more than one version locally and its operation is fully dependent on the availability of the server. An example of centralized VCS is *SubVersioN* (SVN).

The authoritative resource on SVN is *Version Control with Subversion* [22], affectionately known as *the Subversion book*.

By comparison, there is no designated server in decentralized VCS and the users can upload and download new versions directly from one another. The client software is *thick* in the sense that all users have a local repository with every existing version, which they can view and manipulate at any time. The disadvantages include the more complex workflow, greater storage size requirements and the increased opportunity for the users not to share their local changes frequently enough, leading to an increased chance of collisions. Examples of decentralized VCS include Git, Mercurial, or Bazaar.

Although VCS can be used to keep track of any kind of files, they are especially geared towards text files, which they can easily display along with changes. However, most interactive DPSes do not produce text files, which can make version control challenging. As a solution, some DPSes include internal version control func-

After a remote repository has been established, users download the latest version of the document and then keep downloading the latest changes by other users and uploading changes of their own.

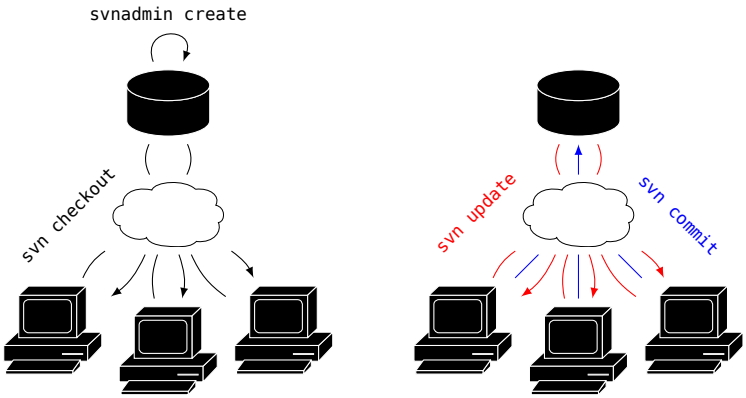
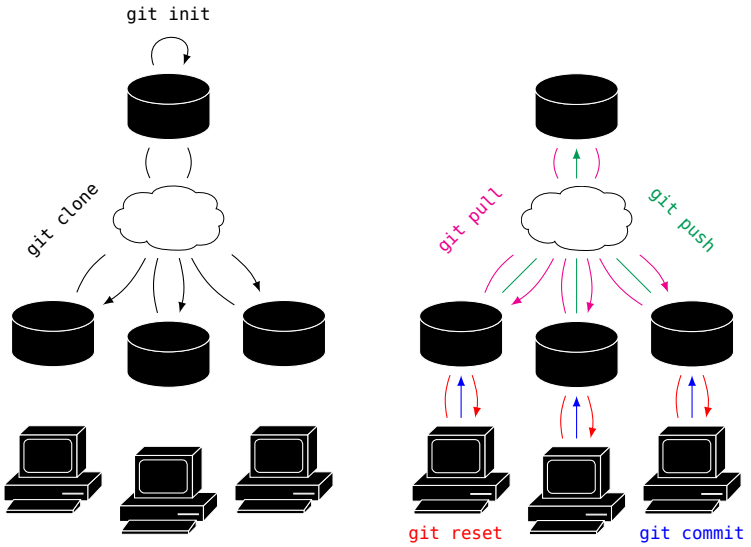


Figure 1.8: The basic SVN workflow

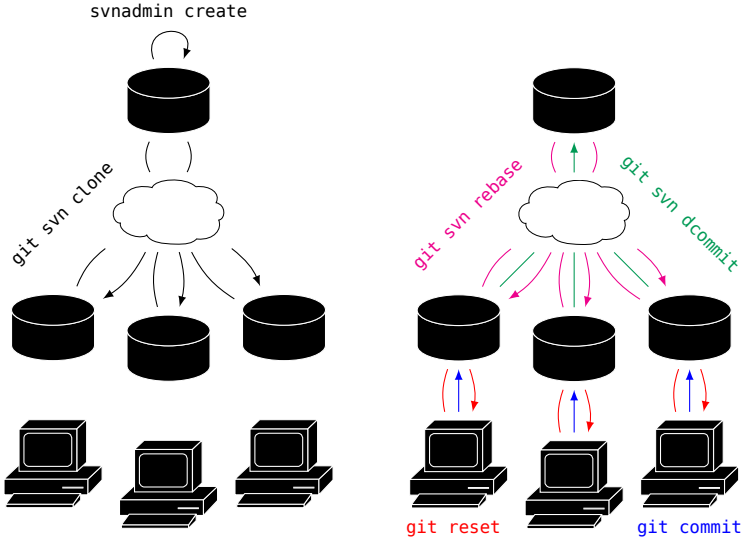
An example would be the graphical SVN client Tortoise SVN that is able to display the changes between two versions of Microsoft Word documents using the interface provided by Microsoft Office.

tionality that can record changes directly into output files. Other DPSes provide an interface for external VCS to display changes between two versions of output documents produced by the DPSes. A category of its own form web services that enable real-time interactive collaboration—such as Word Online or Google Documents.



After a remote repository has been established, users make local copies of the entire repository and then store changes in their local repositories or revert changes from their local repositories. Users periodically download the latest changes by other users and upload changes of their own.

Figure 1.9: The diagram above depicts the basic Git workflow. The diagram below depicts the use of the Git program with an SVN repository; this bears all the advantages and disadvantages associated with decentralized VCS.



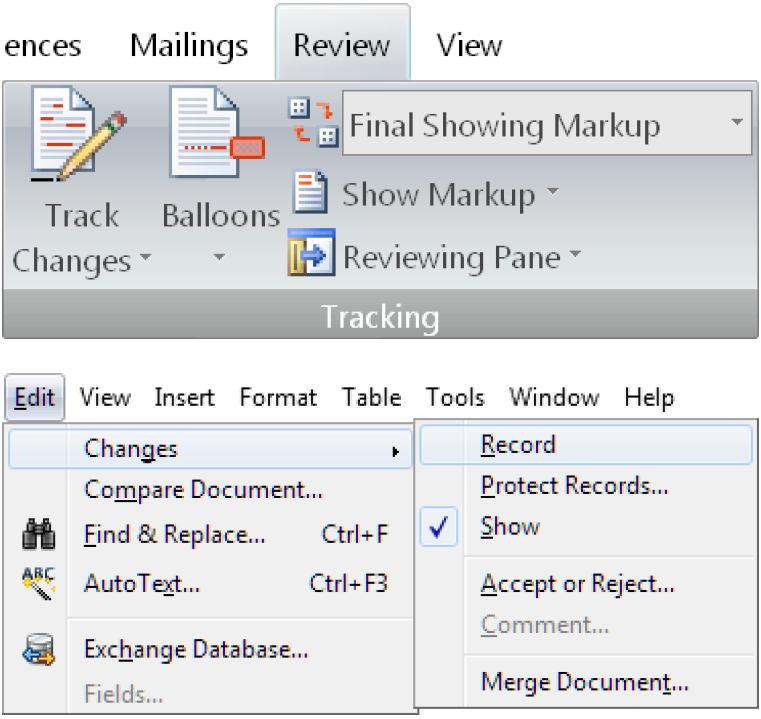


Figure 1.10: The built-in VCS of Microsoft Word (top) and Apache OpenOffice (bottom)

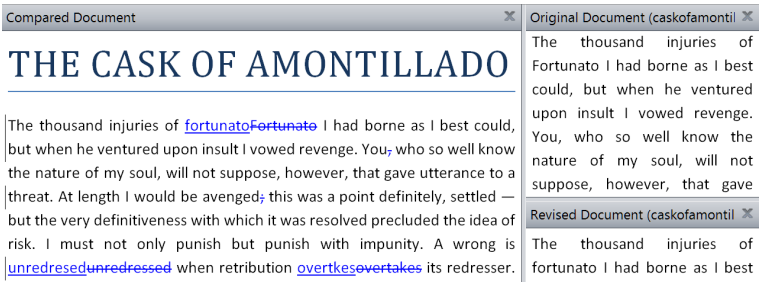


Figure 1.11: Tortoise SVN is a graphical frontend for SVN with the ability to display the difference between two versions of a Microsoft Word document even though it is not a text file.

Chapter 2

Markup

A manuscript can be a seamless current of words and still make perfect sense to an author. To truly capture its meaning in a clear and unambiguous manner, however, the author will often need to supplement the manuscript with a set of annotations. At a more fundamental level, this refers to the compliance with the orthographic rules—such as the correct spelling, capitalization, word breaks, and punctuation—that are specific to the language of the document. It is not at all unreasonable to expect that this basic compliance should be already met by the manuscript. At a higher level, this consists of discovering and marking up the inner order and logic of the text, so that the resulting document can later be typeset in a way that visually reflects its structure.

It is not unusual for an author to write and mark up their manuscript at the same time. Nevertheless, each of the two activities represents a distinct concept. Writing is the process of breaking ideas down into raw sequences of words. To mark up these words then is to take and reassemble them back into meaningful units of linguistic thought.

Markup can be created using a variety of *markup languages*. Aside from *logical markup*, which captures the logical structure of a document, markup languages may also provide *presentation markup*, which directly impacts the visual properties of the document but carries no semantic information. The usage of presentation markup makes it impossible to separate the markup from the design and to capture the structure of the document. As a result,

the consistency in the design of each logical part of the document needs to be ensured manually, and future changes of design become error-prone and tedious. In this regard, logical markup is to design what style guides are to writing: a means of ensuring internal consistency that should be used whenever possible.

2.1 Meta Markup Languages

2.1.1 The General Markup Language

The situation engulfing digital typesetting was growing increasingly frustrating for publishers in the 1960s. The markup languages used by different typesetting systems varied wildly and once a publisher had a large collection of documents typeset via a given company, switching to another one could be a costly venture. This power imbalance artificially increased the price of digital typesetting, leading to a demand for a universal markup language.

More information about the project can be found within *the Roots of SGML – A Personal Recollection* [23] and *SGML: The Reason Why and the First Published Hint* [24].

This demand was met by a project developed at the Cambridge Scientific Center of *the International Business Machines Corporation* (IBM) in the early 1970s. The project aimed at imbuing a text editor with the ability to query, edit, and display documents from a central repository to allow the usage of computers in legal practice. Very early on in the development it became apparent that the main problem were going to be the markup languages in which the documents were written. These languages varied wildly and many of them comprised largely presentation markup, which made information retrieval impossible without heavy use of heuristics. To resolve these issues, a unifying markup language called *the General Markup Language* (GML) was drafted. The language was released [25] to the public in 1981 and finally standardized in 1986 as *the Standard General Markup Language* (SGML). [26]

The authoritative resource on SGML is *the SGML Handbook* [27], which includes the full text of the standard bearing extensive annotations.

SGML documents consist of text mixed with *tags*, which delimit meaningful sections of the document called *elements*. Elements may carry additional information in *attributes*. Additionally, SGML documents may contain miscellaneous instructions for the programs that are processing them as well as human-readable comments. An umbrella term for the various parts of SGML document is *nodes*. Repeated strings of text can be declared as *entities* that can be used throughout the document in place of the original strings.

Although the described structure is shared by all SGML documents, the actual syntax, as well as the restrictions regarding the contents and the attributes of individual elements, are declared within a *Document Type Declaration* (DTD), which can be different for each document. It is worth noting that a DTD only declares the syntax of an SGML document; the semantics of the individual elements and their attributes are left to the interpretation of the program processing the document. The syntax and the constraints imposed by a DTD define an *application of SGML*. An SGML document is considered to be a valid instance of an SGML application, when it conforms to the corresponding DTD.

2.1.2 The Extensible Markup Language

Although SGML was designed to be the general format for data exchange, the complexity of the specification and the lack of support for Unicode (see Section 1.1.1) proved to be a major hindrance preventing its wider adoption and the development of SGML tools. In a response, *the World Wide Web Consortium* (W3C) published a specification of *the eXtensible Markup Language* (XML) [28] in 1998. Along with the introduction of XML, the SGML specification received a technical corrigendum [29], which turned XML into an SGML application defined through a DTD.

This DTD completely fixes the syntax of XML documents, which makes it possible to differentiate between two levels of correctness. An XML document is considered to be *well-formed*, when it conforms to the DTD that specifies the syntax of XML and to the XML specification. An XML document is considered to be *valid* against an DTD, when it is well-formed and conforms to the said DTD. Along with DTDs, there exists a wealth of *schema languages* for XML—such as W3C XML Schema, RELAX NG, or Schematron—that can be used to check the validity of an XML document instead of a DTD. The constrains imposed by either a DTD or a schema define an *application of XML* (also *language* or *format*).

A list of tools for the manipulation of files in XML schema languages is maintained on the Web site of W3C at http://www.w3.org/XML/Schema.

Along with schema languages, other supplementary languages exist, such as XPointer, XPath, and XQuery for the retrieval of data from XML documents, *the Cascading Style Sheets language* (CSS) [30] for the specification of XML document design, and the various languages for the description of Web resources that we will discuss in Section 2.2.3.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE recipe SYSTEM "recipe.dtd">
<recipe>
  <name>Palatschinken</name>
  <description>A Slavic crêpe-like dish</description>
  <ingredientList serves="8">
    <ingredient amount="120g">Plain flour</ingredient>
    <ingredient amount="2">Egg</ingredient>
    <ingredient amount="300ml">Milk</ingredient>
    <ingredient amount="1 tblspn">Oil</ingredient>
    <ingredient amount="1 pinch">Salt</ingredient>
  </ingredientList>
  <stepList>
    <step>Combine the ingredients and whisk until
      you have a smooth batter.</step>
    <step>Heat oil on a pan, pour in a tablespoonful
      of the batter, fry until golden brown.</step>
    <step>Repeat until there is no batter left.</step>
    <step>Serve rolled and filled with jam.</step>
  </stepList>
</recipe>
```

Figure 2.1: An example XML document (`recipe.xml`)

```
<!DOCTYPE recipe PUBLIC "-//EXAMPLE//DTD FOR RECIPES"
  "http://www.example.com/DTD/recipe.dtd">
<!DOCTYPE recipe SYSTEM "recipe.dtd">

<!DOCTYPE recipe [
  <!ELEMENT recipe (name, description, ingredientList,
 ↪  stepList)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT ingredientList (ingredient+)>
  <!ATTLIST ingredientList serves CDATA #REQUIRED>
  <!ELEMENT ingredient (#PCDATA) >
  <!ATTLIST ingredient amount CDATA #REQUIRED>
  <!ELEMENT stepList (step+) >
  <!ELEMENT step (#PCDATA)> ]>

<!DOCTYPE recipe PUBLIC "-//EXAMPLE//DTD FOR RECIPES"
  "http://www.example.com/DTD/recipe.dtd" [
  <!-- Omitted for brevity. --> ]>
<!DOCTYPE recipe SYSTEM "recipe.dtd" [
  <!-- Omitted for brevity. --> ]>
```

DTDs in SGML and XML documents can be either linked to the document through `PUBLIC` and `SYSTEM` identifiers (top), directly embedded in the document (middle), linked to the document and then extended by an embedded specification (bottom), or omitted.

Figure 2.2: An example DTD

```
element recipe {
  element name { text },
  element description { text },
  element ingredientList {
    attribute serves { xsd:positiveInteger },
    element ingredient {
      attribute amount { text }, text
    }+
  }, element stepList {
    element step { text }+
  }
}
```

Figure 2.3: A reformulation of the DTD from Figure 2.2 in the compact syntax of the RELAX NG schema language (`recipe.rnc`). Note how RELAX NG allows us to constrain the attribute data types.

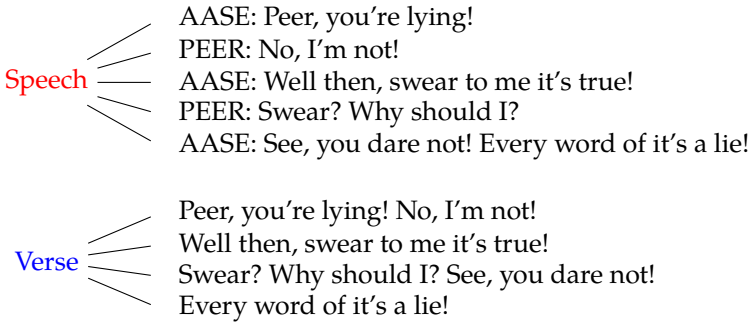
```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="recipe"><complexType><all>
    <element name="name" type="string" minOccurs="1"/>
    <element name="description" type="string"
             minOccurs="1"/>
    <element
 ↪  name="ingredientList"><complexType><sequence>
      <element name="ingredient" minOccurs="1"
               maxOccurs="unbounded">
        <complexType><simpleContent>
          <extension base="string">
            <attribute name="amount" type="string"/>
          </extension>
        </simpleContent></complexType>
      </element></sequence>
      <attribute name="serves" type="positiveInteger"
                 use="required"/>
    </complexType></element>
    <element name="stepList"><complexType><sequence>
      <element name="step" type="string" minOccurs="1"
               maxOccurs="unbounded"/>
    </sequence></complexType></element>
  </all></complexType></element>
</schema>
```

Figure 2.4: A reformulation of the DTD from Figure 2.2 in the XML Schema language (`recipe.xsd`)

```
xmllint -noout --dtdvalid recipe.dtd recipe.xml
xmllint -noout --schema   recipe.xsd recipe.xml
trang recipe.rnc recipe.rng # Compact -> Full Relax NG
xmllint -noout --relaxng  recipe.rng recipe.xml
```

Figure 2.5: XML documents can be easily validated against XML schemata using the free command-line program of `xmllint`.

A notable feature of XML unavailable in SGML are *namespaces*, which were added to the XML specification [32] in 1999. Namespaces enable the inclusion of elements and attributes from different XML applications within a single XML document; each application is uniquely identified through *the Internationalized Resource Identifiers* (IRIS) [33]. Namespaces in XML are a spiritual successor of a more expressive SGML feature of `CONCUR`, which makes it possible to mark up several structural views of a single document. Unlike with `CONCUR`, which ties each view to an SGML DTD, there exists no general mechanism for the translation of the IRIS to XML




```
<(V)line>
  <(S)speech who="Aase">Peer, you're lying!</(S)speech>
  <(S)speech who="Peer">No, I'm not!</(S)speech>
</(V)line><(V)line>
  <(S)speech who="Aase">Well then,
    swear to me it's true!</(S)speech>
</(V)line><(V)line>
  <(S)speech who="Peer">Swear, why should I?</(S)speech>
  <(S)speech who="Aase">See, you dare not!
</(V)line><(V)line>
  Every word of it's a lie!</(S)speech>
</(V)line>
```

Figure 2.6: The markup of the dramatic and metrical views of Henrik Ibsen's *Peer Gynt* using the `CONCUR` feature of SGML. This figure was inspired by the figures found in the article GODDAG: *A Data Structure for Overlapping Hierarchies* [31].

schemata. This makes it impossible to validate namespaced XML documents, unless all the IRIs and their schemata are known to the parser.

Due to the reduced complexity of XML compared to SGML, the language was adopted by the industry and has superseded SGML in most applications. Some of the applications of XML for document preparation include DocBook—a technical documentation markup language used for authoring books by publishers such as O'Reilly Media and for documenting software at companies such as Red Hat, SUSE, or Sun Microsystems—, *the Text Encoding Initiative* (TEI)—a general text encoding markup language for the use in the academic field of digital humanities—, *the Mathematical Markup Language* (MATHML)—a markup language for the description of mathematical formulae—, or *the Scalable Vector Graphics language* (SVG)—a vector graphics format. Other XML applications, such as XHTML and RDF/XML, will be discussed in Section 2.2.

The authoritative resource on the DocBook XML format is *DocBook 5: The Definitive Guide* [34]. The book itself is written in DocBook and its source code is publicly available at http://docbook.org.

2.2 Markup on the World Wide Web

2.2.1 The Hypertext Markup Language

In 1989, an English computer scientist named Timothy John Berners-Lee proposed a decentralized system for sharing documents within *the European Organization for Nuclear Research* (*la Conseil Européen pour la Recherche Nucléaire*, CERN) [35]. The system laid foundation for the Web and earned its author knighthood. The markup language used to write documents for the system was an application of SGML called *the HyperText Markup Language* (HTML). In 1993, the Web started to gain traction among the general public owing largely to the release of the first graphical Web browser Mosaic, which paved way for the Web browsers of today. In 1994, Timothy John Berners-Lee formed W3C, which has since developed the standards for the Web.

The Postel's law states that one should be conservative in what they send, but liberal in what they accept. [37, sec. 2.10] It is one of the base principles for building robust communication protocols.

The first standard version of HTML was HTML 2.0 [36] published in 1995. As the Web was becoming ubiquitous, it began accumulating an increasing number of documents that weren't valid instances of HTML, since most Web browsers faced with a malformed document would act in accordance with the Postel's law and try to render the document despite its deficiencies. In

an attempt to unify the way malformed HTML documents were rendered across the Web browsers, W3C acknowledged and documented this behavior as a part of the HTML5 specification [38, sec. 8.2]. An example of a non-conforming HTML5 document and its canonical interpretation is given in Figure 2.7.

Initially, HTML only comprised a mixture of logical and presentation markup with fixed visual interpretation. This changed with the specification of CSS, which was introduced by W3C in 1996. The language enabled the specification of the visual properties for any HTML element, which enabled the separation of document markup and design, effectively eliminating the need for the presentation markup.

During the same period, an initial version of a scripting language called JavaScript [39] was drafted and incorporated into Netscape Navigator 2.0—one of the contemporary leading web browsers and a descendant of the original Mosaic browser. As a part of a joint effort by Sun Microsystems and Netscape Communications to bring the programming language of Java into web browsers, JavaScript was supposed to complement Java applets [40]—a role it has since outgrown. Standardized in 1997 [39], JavaScript blurred the line between static documents and interactive applications and remains the predominant client-side programming language of the Web. However, since the support of JavaScript by a Web browser is fully optional, it is considered a good practice not to depend on JavaScript for the rendering of HTML documents. In the case of interactive HTML applications, this recommendation may be relaxed.

JScript and VBScript competed directly with JavaScript, but they never saw implementation outside Microsoft browsers.

2.2.2 The Extensible Hypertext Markup Language

Ever since the release of XML in 1998, W3C entertained the idea of turning HTML into an application of XML, rather than of SGML, as

```
<b>Bold, <i>bold and italic</b>, italic.</i>
<b>Bold, </b><i><b>bold and italic</b>, italic.</i>
```

Figure 2.7: The first line contains overlapping elements and, as such, can't be a part of a valid HTML document. Nevertheless, browsers should handle it identically to the second line.

```
<font face="Verdana" size="4">
  <font size="+2"><b>SO WHAT IS THIS ABOUT?</b></font>
  <br><br>There is a continuing need to show the power of
  <i>CSS</i>. The Zen Garden aims to excite, inspire,
  and encourage participation. To begin, view some of the
  existing designs in the list. Clicking on any one will
  load the style sheet into this very page. The <i>HTML
  </i> remains the same, the only thing that has changed
  is the external <i>CSS</i> file. Yes, really.
</font>
```

Figure 2.8: An excerpt from the Web site of the CSS Zen Zarden located at http://csszengarden.com. The document above was created using the HTML presentation markup. The document below achieves the same appearance by the combination of logical markup and CSS.

```
<style>
  body {
    font-family: Verdana;
    font-size: large;
  } h1 {
    font-size: x-large;
    text-transform: uppercase;
  } abbr {
    font-style: italic;
  }
</style>

<h1>So what is this about?</h1>
<p>There is a continuing need to show the power of
<abbr>CSS</abbr>. The Zen Garden aims to excite, inspire,
and encourage participation. To begin, view some of the
existing designs in the list. Clicking on any one will
load the style sheet into this very page. The
<abbr>HTML</abbr> remains the same, the only thing that
has changed is the external <abbr>CSS</abbr> file. Yes,
really.</p>
```

exemplified by the working draft of *Reformulating HTML in XML* [41]. Unlike HTML parsers, whose acceptance of malformed content makes them complex, XML parsers are required to strictly refuse XML documents that aren't well-formed [28, Section 1.2, Terminology], leading to architectural simplicity and decreased computational requirements. As a result, reformulating HTML in XML was suggested as a way to bring the Web to mobile, embedded, and other devices limited in their computational resources and to reduce the amount of malformed documents on the Web in general. Other perceived advantages included the ability to use XML tools for web documents and to include instances of other XML applications—such as MATHML and SVG—directly into web documents through XML namespaces.

The idea was brought to fruition in the XML application of *the eXtensible HyperText Markup Language* (XHTML) [42]. However, the supposed benefits proved to be too marginal to warrant migration from HTML. The speed advantages of the simplified processing were largely offset by the lack of support for incremental rendering, since it is impossible to validate and render partially downloaded XHTML documents and the advances in the area of mobile devices made HTML processing sufficiently fast. The lack of ways to provide alternative content for browsers that would not support the XML applications instantiated in the XHTML documents also reduced the usefulness of the XML namespaces in XHTML considerably. As a result, XHTML has yet to succeed in replacing HTML and remains a minority markup language on the Web.

2.2.3 The Semantic Web and Linked Data

The Web is based on the idea of a distributed and globally available network of human knowledge. The languages of HTML, XHTML, CSS and JavaScript form the foundation of the human-readable parts of the Web, but are inadequate for creating a network of machine-readable data that could be navigated by software agents. Drawing from the research in the field of knowledge representation, W3C created *the Resource Description Framework* (RDF) [44] in 1999—a language for the description of resources on the Web.

The idea of a network of machine-readable data was described by Tim Berners-Lee in 2006 in the article *Linked Data* [43].

An RDF document represents data as a set of *triplets*. Each triplet comprises a *predicate*, a *subject*, and an *object*, where both the predicate and the subject are specified as *resources* using IRIs.

If the object of a triplet (p, s, o) is also a resource, the triplet can be interpreted as a subject s being in a relation p with the object o. If the object is a *literal value* rather than a resource, the triplet can be interpreted as a subject s having a property p with the value o.

Resources in RDF are specified via IRIs to prevent naming collisions in RDF documents created independently by distinct authors. These IRIs do not need to point to any existing web page, and—beside the small set of standard resources specified within the RDF specification—they carry no inherent meaning. In order to describe a set of resources, the relationships between them, and their intended meaning in an RDF document, an extension of the set of standard resources called RDF Schema [45] can be used. The resulting documents are called *ontologies* and can be used for automated reasoning about RDF documents containing resources described by the ontology. Some of the well-known ontologies include *the Dublin Core* (DC)—an ontology for the generic description of resources, both digital and physical—, *Friend Or A Foe* (FOAF)—an ontology for the description of people and their social relationships—, or the Music Ontology—an ontology for the description of entities related to the music industry, such as albums, artists, tracks, and events. More expressive standards for the creation of ontologies, such as *the Web Ontology Language* (OWL) [46], also exist.

A list of ontologies that are fully documented, honor the current best practices, and are supported by various tools can be found on the W3C wiki at http://www.w3.org/wiki/Good_Ontologies.

RDF documents can be represented through many languages, including XML [44], *JSON for LD* (JSON-LD) [47], Turtle [48], and N-Triples [49]. Although RDF documents in any of these representations can be included in or linked to HTML and XHTML documents, this will often result in the undesirable duplication of data. To prevent this, the language of *RDF in attributes* (RDFA) [50] makes it possible to mark parts of the HTML or XHTML document as RDF data. The usage of RDF in conjunction with HTML and XHTML is intended to gradually obsolete the loosely-defined use of HTML and XHTML attributes, the `<meta>` and `<link>` elements, and the CSS class names to include additional machine-readable metadata into the documents on the Web—a technique known as *microformatting*.

2.3 Document Preparation Systems

Some of the existing markup languages are tied directly to specific *Document Preparation Systems* (DPSes). These DPSes can be

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
 ↪  rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/terms/"
         xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description
 ↪  rdf:about="http://example.org/document.html">
    <dc:title xml:lang="en">John's Web page</dc:title>
    <dc:creator
 ↪  rdf:resource="http://example.org/john-smith"/>
  </rdf:Description>
  <rdf:Description
 ↪  rdf:about="http://example.org/john-smith">
    <rdf:type rdf:resource="foaf:Person"/>
    <foaf:name>John Smith</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

```
<http://example.org/document.html>
 ↪  <http://purl.org/dc/terms/title> "John's Web page"@en
<http://example.org/document.html>
 ↪  <http://purl.org/dc/terms/creator>
 ↪  <http://example.org/john-smith>
<http://example.org/john-smith>
 ↪  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
 ↪  <http://xmlns.com/foaf/0.1/Person>
<http://example.org/john-smith>
 ↪  <http://xmlns.com/foaf/0.1/name> "John Smith"
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
<http://example.org/document.html>
    dc:title "John's Web page"@en ;
    dc:creator <http://example.org/john-smith> .
<http://example.org/john-smith>
    a foaf:Person ;
    foaf:name "John Smith" .
```

Figure 2.9: An example RDF document using the DC and FOAF ontologies in the languages of RDF/XML (`john.rd`, top), N-Triples (`john.nt`, middle), and Turtle (`john.ttl`, bottom)

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="meta" type="application/rdf+xml"
          href="john.rdf">
    <link rel="meta" type="text/turtle" href="john.ttl">
    <link rel="meta" type="application/n-triples"
          href="john.nt">
    <title>John's Web page</title>
  </head>
  <body>
    Hi, I'm John Smith.
  </body>
</html>
```

Figure 2.10: Above is an HTML document linked to the RDF document from Figure 2.9. Below is the same HTML document with the RDF data directly embedded using the RDFA language.

```
<!DOCTYPE html>
<html lang="en">
  <head vocab="http://purl.org/dc/terms/"
        about="http://example.org/document.html">
    <title property="title" lang="en">John's Web
 ↪  page</title>
    <meta property="creator"
          href="http://example.org/john-smith">
  </head>
  <body vocab="http://xmlns.com/foaf/0.1/"
        about="http://example.org/john-smith"
        typeof="Person">
    Hi, I'm <span property="name">John Smith</span>.
  </body>
</html>
```

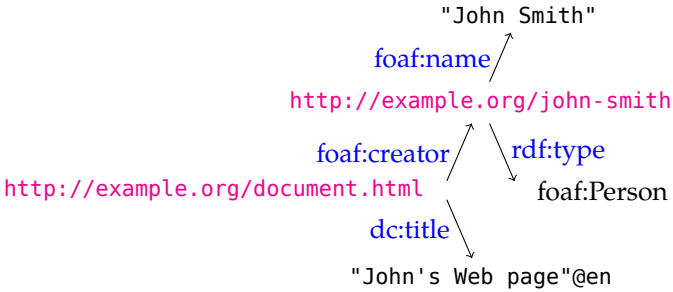


Figure 2.11: A graph of the RDF document in Figure 2.9

categorized into the *batch-oriented*, which process text files into printable output documents on demand, and the *interactive* (also *What You See Is What You Get* (WYSIWYG)), which allow the user to directly edit an approximation of the output document through a visual editor. The price for the mild learning curve of interactive DPSes are the more primitive typesetting algorithms, which need to be sufficiently fast to enable real-time user interaction, and the reduced flexibility stemming from the usage of a *Graphical User Interface* (GUI), which, although often intuitive for simple tasks, seldom matches the power of the markup languages used by batch-oriented DPSes.

2.3.1 Batch-oriented Systems

One of the archetypal batch-oriented DPSes are `troff`, whose function is to produce output for general printers, and `nroff`, whose function is to produce output for line printers and text terminals. Both are proprietary software developed for the Unix operating system at the beginning of 1970s by *the American Telephone and Telegraph corporation* (AT&T). An alternative to `nroff` and `troff` is `groff`, which was developed as free software for the *GNU is Not Unix* (GNU) project in 1980 by the members of the *the Free Software Movement* (FSM). `Groff` combines the capabilities of both systems and is used extensively for the markup of documentation in Unix and Unix-like operating systems. The markup language of `groff` combines presentation markup with programming constructs and enables the definition of logical markup through user macros. The

standard macro packages for `groff` include `man` for the formatting of documentation, `me` for the creation of research papers, and the more recent `mom` for general typesetting tasks. Special markup invokes preprocessors that can be used for the typesetting of tables, equations, and vector graphics.

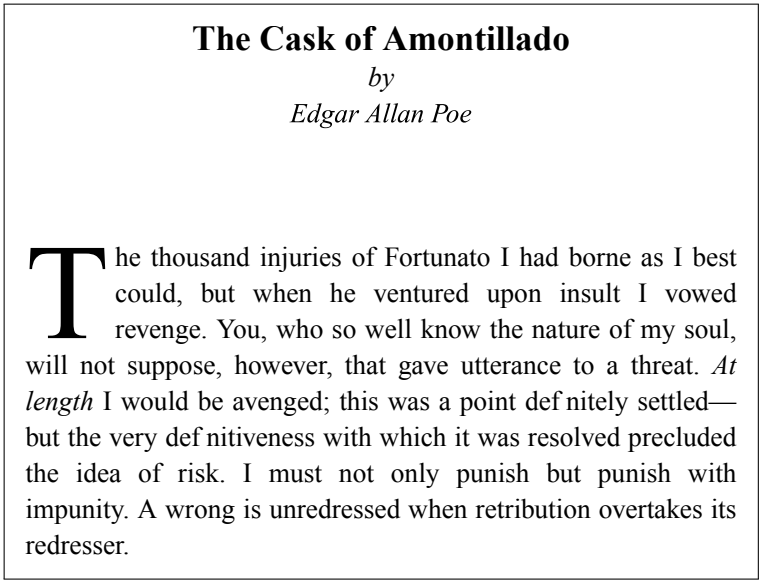
The circumstances that led to the creation of TEX and the surrounding tools are thoroughly documented in *Digital Typography* [52].

Another notable free batch-oriented DPS is TEX, which was developed in the 1970s by an American professor of computer science Donald Knuth after he had received galley proofs for the second volume of his monograph, *the Art of Computer Programming*, and found the appearance of mathematical formulae distasteful. As a result, the typesetting of mathematics is a central theme in TEX, rather than an afterthought, which differentiates it from most other DPSes and which contributes to the massive popularity TEX has enjoyed among academics. Much like in the case of `troff` and its derivatives, the language of TEX contains only typographic and programming primitives, but the creation of logical markup is possible through user macros. A popular TEX macro package that enables the creation of various types of documents with just logical markup is LATEX: the standard markup language for academic and technical documents.

2.3.2 Interactive Systems

Interactive DPSes come in two distinct flavors. Word processors are the digital progeny of the typewriter machine, whose output documents served as manuscripts to be typeset by a typographer. With the advent of personal computing and the Web, self-publishing became more affordable to the general public and modern word processors can be used not only to write but also to design and typeset documents, although the offered functionally is typically limited to ensure ease of use. This concern is not shared by *DeskTop Publishing (DTP) software*, which provides refined control over the resulting page layout and the typesetting at the expense of a steeper learning curve.

Most interactive DPSes will provide a means to mark up sections of text. Presentation markup enables direct changes to the design, whereas logical markup enables the classification of sections of text with the ability to set up the design of each class later on. This decouples writing and markup from design and makes it easy to consistently change the design of an entire document.



The Cask of Amontillado

by

Edgar Allan Poe

The thousand injuries of Fortunato I had borne as I best could, but when he ventured upon insult I vowed revenge. You, who so well know the nature of my soul, will not suppose, however, that gave utterance to a threat. *At length* I would be avenged; this was a point def nitely settled—but the very def nitiveness with which it was resolved precluded the idea of risk. I must not only punish but punish with impunity. A wrong is unredressed when retribution overtakes its redresser.

```
.TITLE     "The Cask of Amontillado
.AUTHOR    "Edgar Allan Poe
.PRINTSTYLE TYPESET
.PAGE 6i 9i .75i .75i .75i .75i
.START
.PP
.DROPCAP T 3
he thousand injuries of Fortunato I had borne as I best
could, but when he ventured upon insult I vowed revenge.
You, who so well know the nature of my soul, will not
suppose, however, that gave utterance to a threat.
\*[IT]At length\*[PREV] I would be avenged; this was a
point definitely settled\[em]but the very definitiveness
with which it was resolved precluded the idea of risk. I
must not only punish but punish with impunity. A wrong is
unredressed when retribution overtakes its redresser.
```

Figure 2.12: An excerpt from the beginning of Edgar Allan Poe's *Cask of Amontillado* as a text marked up using the `mom` macro package of `groff` (below) and the output document (above). The marked up text was borrowed from the web page of `mom` [51].

```
% Page geometry
\pdfpagewidth=6in              \pdfpageheight=9in

% Page dimensions
\hsize=\dimexpr\pdfpagewidth-1.5in
\vsize=\dimexpr\pdfpageheight-1.5in
\baselineskip=16.8pt
\hoffset=-.25in                \voffset=-.25in

% Fonts
\font\rm=ptmr8t at 12.5pt\rm   \font\bigbf=ptmb8t at 16pt
\font\dropcap=ptmr8t at 62pt   \font\it=ptmri8r at 12.5pt

% Logical markup definition
\def\title#1{{\bigbf\centerline{#1}}}
\def\author#1{{\it\centerline{by}\centerline{#1}}%
  \vskip 3.9em}
\def\chapter#1{\noindent\smash{\hskip0.1ex\lower5.8ex%
  \hbox{\llap{\dropcap#1}}\hskip-0.3ex}
  \parshape=4 3em\dimexpr\hsize-3em 3.28em
                \dimexpr\hsize-3.28em 3.28em
                \dimexpr\hsize-3.28em 0em\hsize}

% The document
\title{The Cask of Amontillado}
\author{Edgar Allan Poe}
\chapter The thousand injuries of Fortunato I had borne
as I best could, but when he ventured upon insult I vowed
revenge. You, who so well know the nature of my soul,
will not suppose, however, that gave utterance to a
threat. {\it At length} I would be avenged; this was a
point definitely settled---but the very definitiveness
with which it was resolved precluded the idea of risk. I
must not only punish but punish with impunity. A wrong is
unredressed when retribution overtakes its redresser.\bye
```

Figure 2.13: The document from Figure 2.12 reformulated in TEX using plain TEX macros and the primitives of ε-TEX and pdfTEX

At length I would be avenged; this was a point definitely settled

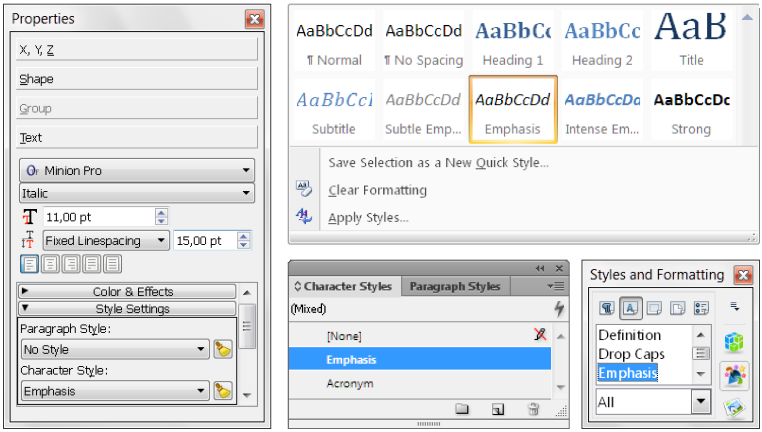


Figure 2.14: Logical markup in the interactive DPSes of Scribus (left), Microsoft Word (top), Adobe InDesign (bottom left) and Apache OpenOffice (bottom right)

2.4 Lightweight Markup Languages

Parallel to the heavy-duty applications of SGML and XML, there runs a vein of markup languages that give priority to unobtrusiveness and legibility over raw expressive power. Rooted in the reality of computer text terminals with limited formatting capabilities, *lightweight markup languages* leverage punctuation and indentation to produce comparatively weak and domain-specific, but also humane, highly intuitive, and often profoundly beautiful markup that is easy to both read and write. Examples of lightweight markup languages include Markdown, Creole, AsciiDoc, MakeDoc, Setext, and Wikicode. Lightweight markup languages are typically supplemented by tools that enable the conversion to more general markup languages, such as HTML. The more popular lightweight markup languages come in various flavors that represent their use cases.

Chapter 3

Design

After a manuscript has been written and marked up, it is time to create a visual system that will emphasize the internal structure and the character of the document. In print design, this involves the selection of one or several typefaces that are well-suited to both the document and each other, the design and the positioning of the structural elements of the document—such as headings, tables, figures, and lists, and the choice of the paper size and the page layout. In web design and multi-target publishing, several visual systems may have to be created to accommodate for various display devices.

3.1 Fonts

When choosing *typefaces* for a document, legibility should be of foremost concern. The *body text* should be set with a typeface at a size of at least 10 pt, if the document is aimed at adult readers, or 12 pt, if visually impaired readers and elementary-school students are a part of the audience. [53, para. 13–15]. The target medium also needs to be taken into consideration. A faithful copy of a typeface designed for the letterpress will look lighter than originally intended when printed digitally. This may hamper its legibility, if it contains hairline strokes [54, sec. 6.1.2]. In printed documents, typefaces with *serifs* are more familiar to the reader and therefore more suitable for long-distance reading than their *sans serif* coun-

terparts. At low-resolution screens, however, simple low-contrast typefaces with slab or no serifs will often yield the best result.

A typeface should also contain all the letters and symbols that will appear in the document. If the manuscript is multilingual and contains passages in both Latin and non-Latin writing systems, it may be necessary to combine several typefaces. If the multilingual manuscript only contains Latin characters, but several accented characters are missing from the body text typeface, they may be constructed by combining the body text typeface with diacritical marks from another font family. If certain punctuation marks and other symbols are missing from the body text typeface, they may likewise be borrowed from other font families. The typefaces should be consonant in their spirit and structure, unless the text would benefit from the dissonance. [54, sec. 5.1.2]

Beside the body text typeface, several other typefaces may appear in a document—a bold face, an italic face, or perhaps several sizes of the body text typeface for use in the structural elements. The natural instinct is to pick these typefaces from a single font family, but some families may not offer all typefaces that the design requires. In these cases, the typefaces may again have to be borrowed from other font families.

3.2 Structural Elements

3.2.1 Paragraphs and Stanzas

As the base units of linguistic thought in prose, *paragraphs* split the text into coherent portions ready for consumption. A line in a paragraph of the body text should be 45–75 characters long on a single-column page or 40–50 characters long on a multi-column page and *justified* (spread horizontally to fit the column width). Extended passages of lines wider than 80 characters strain the eye of the reader, whereas justified lines that are too narrow to accommodate 40 characters may make the word spacing entirely too loose. In the latter case, the text should be set *ragged* instead, as seen in the sidenotes throughout this book [54, sec. 2.1.2].

Vertically, the lines of a paragraph should be separated by approximately twenty to forty-five percent of the typeface size [55]. If the size of the body text typeface is 10 pt, then the body text

> The second function of Soul – knowing – was not at first distinguished from motion. Aristotle says, φαμὲν γὰρ τὴν ψυχὴν λυπεῖσθαι χαίρειν, θαρρεῖν φοβεῖσθαι, ἔτι δὲ ὀργίζεσθαί τε καὶ αἰσθάνεσθαι καὶ διανοεῖσθαι· ταῦτα δὲ πάντα κινήσεις εἶναι δοκοῦσιν. ὅθεν οἰηθείη τις ἂν αὐτὴν κινεῖσθαι. "The soul is said to feel pain and joy, confidence and fear, and again to be angry, to perceive, and to think; and all these states are held to be movements, which might lead one to suppose that soul itself is moved."

```
\documentclass[11pt]{article}
\usepackage{fontspec, leading, newunicodechar}
\usepackage[Latin, Greek]{ucharclasses}
\setTransitionsForLatin{%
  \fontspec{AlegreyaSans-Regular.ttf}[Ligatures=TeX]}{}
\setTransitionsForGreek{%
  \fontspec{GFSNeohellenic.otf}[Scale=1.2, WordSpace=0.5,
  ↪ Ligatures=TeX]}{}
\newunicodechar{·}{\raisebox{.8ex}{.}}
\frenchspacing
\leading{14pt}

\begin{document}
  The second function of Soul -- knowing -- was not at
  first distinguished from motion. Aristotle says, φαμὲν
  γὰρ τὴν ψυχὴν λυπεῖσθαι χαίρειν, θαρρεῖν φοβεῖσθαι, ἔτι
  δὲ ὀργίζεσθαί τε καὶ αἰσθάνεσθαι καὶ διανοεῖσθαι· ταῦτα
  δὲ πάντα κινήσεις εἶναι δοκοῦσιν. ὅθεν οἰηθείη τις ἂν
  αὐτὴν κινεῖσθαι.
  ``The soul is said to feel pain and joy, confidence and
  fear, and again to be angry, to perceive, and to think;
  and all these states are held to be movements, which
  might lead one to suppose that soul itself is moved.''
\end{document}
```

Figure 3.1: An excerpt from F. M. Cornford's *From Religion to Philosophy: A Study in the Origins of Western Speculation* as a text marked up in TEX using LATEX macros and the primitives of XETEX (below) and the output document (above). Note that two typefaces were used: the regular typeface of Alegreya Sans at the size of 11 pt for the Latin characters and the regular typeface of GFS Neohellenic at the size of 13.2 pt for the Greek characters.

```
<style>
  @font-face {
    font-family: "Alegreya Sans";
    src: url("AlegreyaSans-Regular.ttf");
 ↪  format("truetype");
    unicode-range: U+00-24F, U+1E00-1EFF, U+2000-206F,
 ↪  U+2C60-2C7F, U+A720-A7FF, U+FB00-FB4F;
  } @font-face {
    font-family: "GFS Neohellenic";
    src: url("GFSNeohellenic.otf"); format("opentype");
    unicode-range: U+2C80-2CFF, U+370-3FF, U+1F00-1FFF,
 ↪  U+102E0-102FF;
  } p {
    font-family: "Alegreya Sans", "GFS Neohellenic",
 ↪  sans-serif;
    line-height: 14pt;
  } [lang="en"] {
    font-size: 11pt;
  } [lang="gr"] {
    font-size: 13.2pt;
  }
</style>

<p><span lang="en">The second function of Soul —
knowing — was not at first distinguished from motion.
Aristotle says, </span><span lang="gr">φαμὲν γὰρ τὴν
ψυχὴν λυπεῖσθαι χαίρειν, θαρρεῖν φοβεῖσθαι, ἔτι δὲ
ὀργίζεσθαί τε καὶ αἰσθάνεσθαι καὶ διανοεῖσθαι· ταῦτα δὲ
πάντα κινήσεις εἶναι δοκοῦσιν. ὅθεν οἰηθείη τις ἂν αὐτὴν
κινεῖσθαι. </span><span lang="en">“The soul is said to
feel pain and joy, confidence and fear, and again to be
angry, to perceive, and to think; and all these states
are held to be movements, which might lead one to suppose
that soul itself is moved.”</span></p>
```

Figure 3.2: The document from Figure 3.1 reformulated in HTML5 and CSS3

line height (also known as the *leading*) would be between 12 and 14.5 pt, adding 1 to 2.25 pt of lead above and below each line. As a general guideline, dark and bulky typefaces require more leading, as do texts riddled with accents, full capital letters, subscripts, and superscripts [54, sec. 2.2.1]. The body text of this book is set in 10 pt Palatino with the leading of 12 pt. To allow for such minimal leading, all acronyms and other strings of upper-case letters are set as *small capitals* (capital letters whose height matches the lower case).

Two adjacent paragraphs should be visibly separated without distracting the reader from the text. A predominant method is to indent the initial line of a paragraph with one half (1 en) to three times (3 em) the typeface size. The indent is unnecessary when there is no ambiguity—such as in the first paragraph following a heading. [54, sec. 2.3]

If the margins are ample, outdented paragraphs are an intriguing option as well. ¶ Paragraphs can also be separated by graphical symbols, such as pilcrows, bullets, or boxes. A plain horizontal space that is at least 3 em wide can likewise act as a paragraph separator. [56, ch. 2, p. 16]

Block paragraphs exchange indentation and horizontal separators for additional vertical space above and below the paragraph. In justified block paragraphs, this space can be omitted as well, although the typesetter then has to manually ensure that the last line of each paragraph offers enough horizontal space to act as a separator. In short documents and limited spans of text, block paragraphs are an attractive option. [54, sec. 2.3.2]

Being the verse counterpart to the paragraph, the stanza is a collection of lines rather than of sentences. Due to this structural difference, stanzas are typically only justified, when the individual lines are long enough to fill up the column, and ragged otherwise. Much like in the case of prose, short-form poetry benefits from having the stanzas set in block paragraph style.

3.2.2 Headings

Another fundamental structural element is the *heading*. The function of a heading is to delimit and name the individual sections of a document. To alleviate navigation, headings should be a prominent presence on a page. This can be achieved by using a larger

	Sizes in inches	Page proportions	
A4	8.27×11.7	$2 : \sqrt{2}$	1.41421
B5	6.93×9.84	$1 : \sqrt{2}$	0.707
Letter	$8\frac{1}{2} \times 11$	$1 : 1.294$	1.2941

Table 3.1: An overview of common paper sizes used for commercial and industrial printing

variant of the body text typeface or by including the text of the latest heading in the margin or the header of the page [54, sec. 4.2.1], as seen throughout this book.

The hierarchy of the headings can be expressed through the variation of typefaces, indentation, alignment and numbering, although alternating the size of the body text typeface is sufficient for many types of documents. In documents that are bound in codex form and read two pages at a time, the height of headings should be a whole multiple of the line height of the body text, so that the headings do not disrupt the alignment of lines on the facing pages. [53, para. 33]

3.2.3 Tables and Lists

Tables and *lists* are structural elements that should fit seamlessly into the surrounding text and avoid unnecessary visual clutter. Use the same typeface the surrounding text does, treat the columns of tables the same way you treat columns in the text, and keep the amount of rules, boxes, dots, and extraneous spacing to a bare minimum (see Table 3.1). [54, sec. 2.1.10 and 4.4]

3.2.4 Notes

Notes provide commentary on a specified passage of the main text and can take three different forms:

This is a sidenote. Sidenotes enliven the page and are easy for the reader to find.

1 *Sidenotes* are displayed in the horizontal margins next to the relevant passage of the main text, as seen throughout this book. Unless the horizontal margins are very wide, sidenotes are unsuitable for the inclusion of bibliographical references—a common use for notes in academic writing.

2 *Footnotes* are delegated to the bottom of the page and linked to the relevant passage of the main text through symbols or superscript numbers.[1] Compared to side notes, they are more difficult for the reader to find. Footnotes should align with the bottom of the text block, not stick out into the bottom margin. [53, para. 48]

3 *Endnotes* are delegated to the end of a section or the entire document and are linked to the relevant passage of the body text through superscript numbers. They are the easiest of the three to typeset, but also the hardest for the reader to find.

Notes are typically typeset in sizes from 8 pt up to the body text typeface size depending on their frequency, importance, and average length. [54, sec. 4.3] If several categories of notes are present in the document, it may be desirable to give each a different form.

3.2.5 Quotations

Quotations repeat what has already been expressed somewhere else before and can take two different forms: [54, sec. 5.4]

1 *Run-in quotations* are included directly into the paragraph and set off from the surrounding text using quotation marks in accordance with the orthographic rules on the use of punctuation in the language of the paragraph: "Jesters do oft prove prophets." From the designer's viewpoint, run-in quotations require no special treatment, although it is crucial that the body text typeface contains the required quotation marks.

2 *Block quotations* are set as block paragraphs that are clearly separated from the surrounding text. This involves adding a vertical space above and below the block paragraphs and optionally also changing the typeface, its size, or the indentation of the paragraphs: [54, sec. 2.3.3]

This is the excellent foppery of the world that when we are sick in fortune—often the surfeit of our own behavior—we make guilty of our disasters the sun, the moon, and the stars, as if we were villains by necessity, fools by heavenly compulsion, knaves, thieves, and treachers by spherical predominance, drunkards, liars, and adulterers by an enforced

1 This is a footnote. Due to their width, footnotes can comfortably accommodate full bibliographical references, which makes them popular in academic writing.

A footnote can also contain multiple paragraphs of text, although long footnotes are tedious to read if the size of the typeface is small. [54, sec. 4.3.1]

obedience of planetary influence, and all that we are evil in by a divine thrusting-on. An admirable evasion of whoremaster man, to lay his goatish disposition to the charge of a star!

—William Shakespeare, *King Lear*

Block quotations are ideal for longer quotations and for quotations that should carry more weight that run-in quotations.

3.3 Page Layout

The page consists of a textblock surrounded by margins. The text width area is largely determined by the number of columns and the body text size—as described in Section 3.2.1—as well as by our plans for the horizontal margins. A margin containing an occasional sidenote will require less space that a margin ripe with photographs, tables, and diagrams.

The vertical margins may contain additional navigational aids, such as the page numbers and running headers in this book. [54, sec. 8.5.2]

In print design—and wherever else the page height is fixed—we need to also decide on the text height. The text height needs to be a multiple of the body text line height, so that it is possible to completely fill the text block with text. It is typical to derive the text height from the text width to achieve proportions that work well with the proportions of the page. [54, sec. 8.4.2]

3.4 Color

Three types of *cone cells*—S, M, and L—exist in the human eye, and each type is sensitive to a different range of the *visible light*. The perception of the visible light is what is referred to as *color*. In this section, we will briefly overview the color theory as well as general guidelines with respect to color scheme design.

3.4.1 Theory

In 1932, *the International Commission on Illumination* (*Commission Internationale de l'Éclairage*, CIE) defined the CIE *XYZ color space*,

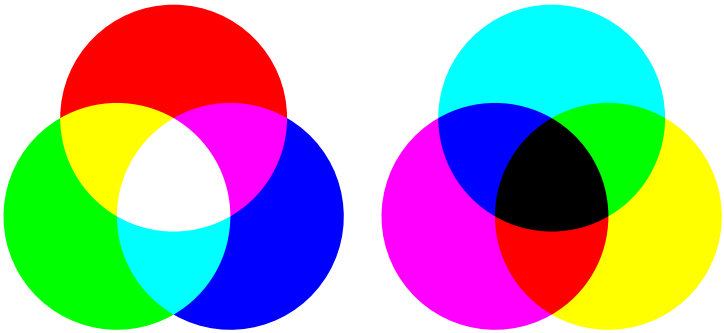


Figure 3.3: The additive *RGB* color model (left), and the subtractive *CMY* color model (right)

which specifies how a *Spectral Power Distribution* (SPD) of the visible light relates to a set of three parameters (X, Y, and Z) that specify a color. [57] Since the parameter Y is a measure of the *luminance* of a color, a two-dimensional color space called the *CIE xyY color space* can by derived from CIE *XYZ* by disregarding the luminosity. A plot, where a point (x, y) corresponds to a color in CIE *xyY*, is called a *chromacity diagram* (see Figure 3.4). In theory, we can directly use the CIE color spaces to specify colors. In practice, there exist task-specific color spaces.

For an approachable introduction to the color theory, see *the Reproduction of Colour* by Hunt [58]. For a dauntingly complete treatment of the art and science of color, see Günther and Styles' *Color Science* [59].

Electronic screens use red, green, and blue sub-pixels. This gives rise to the additive *RGB color model* (see Figure 3.3). An *RGB color space* is then specified by the chromacities of its *primary colors* (*primaries*), and by the *white point*. The extent of the colors that can be represented by an *RGB* color space (its *gamut*) can be plotted in the chromacity diagram as a triangle whose vertices are the chromacities of the primaries. Standard *RGB* color spaces include sRGB, Adobe RGB 1998, and ProPhoto RGB (see Figure 3.4). There exists no single standard white point or primaries; if only the *RGB* coordinates of a color are given, the color cannot be accurately reproduced. For this reason, digital images often include an ICC *color profile*, which fully specifies a color space.

Mixing cyan, magenta, and yellow inks gives rise to the subtractive *CMYK color model*. Although three colors (C, M, and Y) are

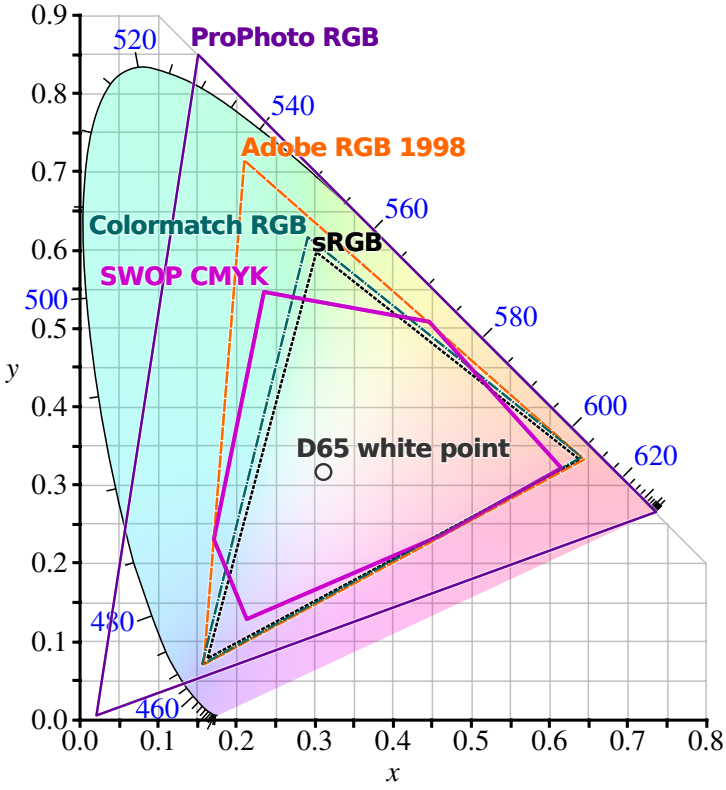


Figure 3.4: A CIE *xyY* chromacity diagram, where the shark-fin-shaped *spectral locus* corresponds to a narrow-band SPD with power at just a single wavelength swept across the visible light (380–700 nm). Source: BenRG and cmglee at Wikimedia Commons.

sufficient to describe a color, a black ink (*K*) is less expensive and dries more quickly than a combination of three colored inks. White color is provided by the paper. Similarly to the *RGB* model, *CMYK* coordinates specify a color only if we specify a *CMYK* color space using an ICC color profile. For printing text, it is often advantageous *not* to specify a color profile, so that no inks other than *K* are introduced in a *color conversion*. When an accurate reproduction of a color is desired, a color model must be specified.

3.4.2 Schemes

In both print and web design, it is perfectly reasonable to use either just the combination of black and white or shades of gray. A secondary color may be introduced to enliven the page if necessary; **red** has historically been used for this purpose (see Figure 3.5). More than a single hue of color may be introduced, although each additional one makes it more difficult to establish a visual system that is intelligible to the reader.

You should only use colored typefaces for emphasis, not for the body text, and on backgrounds that are white or of sufficient contrast with the typeface color. Distinct colors should stay distinct even for the color-blind reader, unless the lack of distinction between the colors does not impair understanding.

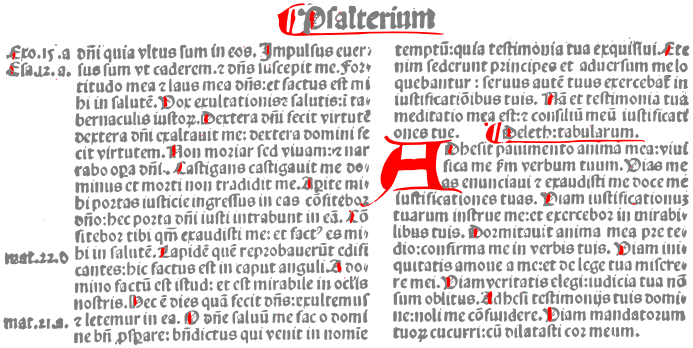


Figure 3.5: An excerpt from the *Latin Vulgate Bible* printed by the German goldsmith, printer, and publisher Anton Koberger in 1487

Bibliography

[1] Mary Brandel. ''1963: The debut of ASCII''. In: *Computerworld* (July 1999). URL: http://edition.cnn.com/TECH/computing/9907/06/1963.idg (visited on 09/06/2015) (cit. on p. 5).

[2] ASA Sectional Committee on Computers and Information Processing. *American Standard Code for Information Interchange*. X 3.4-1963. 10 East 40th Street, New York 16, NY, USA: the American Standard Association, June 1963. URL: http://worldpowersystems.com/J/codes/X3.4-1963/ (visited on 01/28/2015) (cit. on p. 5).

[3] ISO TC97/SC2. *Information technology – ISO 7-bit coded character set for information interchange*. ISO 646:1972. Geneva, Switzerland: the International Organization for Standardization, 1972 (cit. on pp. 5, 7).

[4] ASA Sectional Committee on Computers and Information Processing. *American Standard Code for Information Interchange*. X 3.4-1986. 10 East 40th Street, New York 16, NY, USA: the American Standard Association, June 1986 (cit. on p. 6).

[5] Unicode Consortium. *the Unicode Standard, Version 1.0*. Vol. 1. Reading, MA, USA: Addison-Wesley Developers Press, Oct. 1991. ISBN: 0201567881 (cit. on p. 8).

[6] Unicode Consortium. *the Unicode Standard, Version 1.0*. Vol. 2. Reading, MA, USA: Addison-Wesley Developers Press, June 1992. ISBN: 0201608456 (cit. on p. 8).

[7] ISO/IEC JTC1/SC2. *Information technology – the Universal multiple-octet coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*. ISO/IEC 10646-1:1993. Geneva, Switzerland: the International Organization for Standardization, May 1993 (cit. on p. 8).

[8] ISO/IEC JTC1/SC2. *Transformation Format for 16 planes of group 00 (UTF-16)*. ISO/IEC 10646-1:1993/Amd 1:1996. Geneva, Switzerland: the International Organization for Standardization, Oct. 1996 (cit. on p. 8).

[9] ISO/IEC JTC1/SC2. *UCS Transformation Format 8 (UTF-8)*. ISO/IEC 10646-1:1993/Amd 2:1996. Geneva, Switzerland: the International Organization for Standardization, Oct. 1996 (cit. on p. 8).

[10] Unicode Consortium. *the Unicode Standard, Version 9.0 – Core Specification*. Tech. rep. Mountain View, CA, USA, July 2016. URL: http://www.unicode.org/versions/Unicode9.0.0/UnicodeStandard-9.0.pdf (visited on 09/17/2015) (cit. on pp. 8–10).

[11] Q-Success. *Usage of character encodings for websites*. URL: http://w3techs.com/technologies/overview/character_encoding/all (visited on 09/10/2015) (cit. on p. 9).

[12] Unicode Consortium. *Unicode Technical Standard #10, Version 9.0.0: Unicode Collation Algorithm*. Tech. rep. May 2016. URL: http://www.unicode.org/reports/tr10/tr10-34.html (visited on 09/17/2016) (cit. on p. 10).

[13] Unicode Consortium. *Unicode CLDR Project*. Tech. rep. URL: http://cldr.unicode.org (visited on 09/17/2016) (cit. on p. 10).

[14] ISO TC171/SC2. *Document management – Portable document format*. ISO 32000:2008. Geneva, Switzerland: the International Organization for Standardization, July 2008 (cit. on p. 13).

[15] ISO/IEC JTC1/SC34. *Document description and processing languages – Office Open XML File Formats*. ISO/IEC 29500:2012. Geneva, Switzerland: the International Organization for Standardization, Oct. 2012 (cit. on p. 13).

[16] ISO/IEC JTC1/SC34. *Information technology – Open Document Format for Office Applications (OpenDocument) v1.0*. ISO/IEC 26300:2006. Geneva, Switzerland: the International Organization for Standardization, Dec. 2006 (cit. on p. 13).

[17] Noam Chomsky. ''Three models for the description of language''. In: *Information Theory, IEEE Transactions on* 2.3 (1956), pp. 113–124 (cit. on p. 14).

[18] ISO/IEC JTC1/SC22. *Information technology – the Portable Operating System Interface – Part 2: Shell and Utilities*. ISO/IEC 9945-2:1993. Geneva, Switzerland: the International Organization for Standardization, Dec. 1993 (cit. on p. 14).

[19] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. 3rd ed. O'Reilly Media, 2006, p. 544. ISBN: 9780596528126 (cit. on p. 14).

[20] Unicode Consortium. *Unicode Technical Standard #18, Version 17: Unicode Regular Expressions*. Tech. rep. Nov. 2013. URL: http://www.unicode.org/reports/tr18/tr18-17.html (visited on 09/26/2015) (cit. on p. 16).

[21] Dale Dougherty and Arnold Robbins. *Sed & awk*. Second Edition. O'Reilly Media, 1997. ISBN: 1565922255. URL: http://docstore.mik.ua/orelly/unix/sedawk/ (visited on 09/26/2015) (cit. on p. 16).

[22] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly, 2002. URL: http://svnbook.red-bean.com/ (visited on 09/26/2015) (cit. on p. 17).

[23] Charles F. Goldfarb. ''the Roots of SGML – A Personal Recollection''. In: (1996). URL: http://www.sgmlsource.com/history/roots.htm (visited on 07/29/2015) (cit. on p. 22).

[24] Charles F. Goldfarb. ''SGML: The Reason Why and the First Published Hint''. In: *Journal of the American Society for Information Science* 48 (7 July 1997). URL: http://www.sgmlsource.com/history/jasis.htm (visited on 07/29/2015) (cit. on p. 22).

[25] Charles F. Goldfarb. ''Introduction to Generalized Markup''. In: (1981). URL: http://www.sgmlsource.com/history/AnnexA.htm (visited on 07/29/2015) (cit. on p. 22).

[26] ISO/IECJTC1/SC34. *Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*. ISO/IEC 8879:1986. Geneva, Switzerland: the International Organization for Standardization, Oct. 1986 (cit. on p. 22).

[27] Charles F. Goldfarb. *the SGML Handbook*. New York, NY, USA: Oxford University Press, Inc., 1990. ISBN: 9780198537373 (cit. on p. 22).

[28] Jean Paoli, Tim Bray, and Michael Sperberg-McQueen. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. W3C, Feb. 1998. URL: http://www.w3.org/TR/1998/REC-xml-19980210 (visited on 07/31/2015) (cit. on pp. 23, 31).

[29] ISO/IEC JTC1/SC18/WG8. *Proposed TC for Web SGML Adaptations for SGML*. ISO/IEC N1929. the International Organization for Standardization, June 1997. URL: http://xml.coverpages.org/wg8-n1929-g.html (visited on 07/31/2015) (cit. on p. 23).

[30] Håkon Wium Lie and Bert Bos. *Cascading Style Sheets, level 1*. Recommendation. W3C, Dec. 1996. URL: http://www.w3.org/TR/REC-CSS1-961217 (visited on 07/31/2015) (cit. on pp. 23, 29).

[31] C. M. Sperberg-McQueen and Claus Huitfeldt. "GODDAG: A Data Structure for Overlapping Hierarchies". In: *Digital Documents: Systems and Principles: 8th International Conference on Digital Documents and Electronic Publishing, DDEP 2000, 5th International Workshop on the Principles of Digital Document Processing, PODDP 2000, Munich, Germany, September 13-15, 2000. Revised Papers*. Ed. by Peter King and Ethan V. Munson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 139–160. ISBN: 9783540399162. DOI: 10.1007/978-3-540-39916-2_12 (cit. on p. 27).

[32] Tim Bray, Dave Hollander, and Andrew Layman. *Namespaces in XML*. W3C Recommendation. W3C, Jan. 1999. URL: http://www.w3.org/TR/1999/REC-xml-names-19990114/ (visited on 08/21/2015) (cit. on p. 27).

[33] M. Duerst. *the Internationalized Resource Identifiers (IRIS)*. RFC 3987. RFC Editor, Jan. 2005. URL: http://tools.ietf.org/html/rfc3987 (visited on 08/31/2015) (cit. on p. 27).

[34] Norman Walsh. *DocBook 5: The Definitive Guide*. Apr. 2010. URL: http://www.docbook.org/tdg/en/html/docbook.html (visited on 08/18/2015) (cit. on p. 28).

[35] Tim Berners-Lee. *Information Management: A Proposal*. Tech. rep. Mar. 1989. URL: http://www.w3.org/History/1989/proposal.html (visited on 08/31/2015) (cit. on p. 28).

[36] T. Berners-Lee. *Hypertext Markup Language – 2.0*. RFC 1866. RFC Editor, Nov. 1995. URL: http://tools.ietf.org/html/rfc1866 (visited on 07/31/2015) (cit. on p. 28).

[37] Jon Postel. *DoD standard Transmission Control Protocol*. RFC 761. RFC Editor, Jan. 1980. URL: http://tools.ietf.org/html/rfc761 (visited on 09/16/2016) (cit. on p. 28).

[38] Ian Hickson et al. *HTML5: A vocabulary and associated APIs for HTML and XHTML*. Recommendation. W3C, Oct. 2014. URL: http://www.w3.org/TR/2014/REC-html5-20141028/ (visited on 07/31/2015) (cit. on p. 29).

[39] ECMA International. *Standard ECMA-262 - ECMAScript Language Specification*. Tech. rep. June 1997. URL: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf (visited on 07/31/2015) (cit. on p. 29).

[40] Netscape Communications. *Netscape and Sun announce JavaScript, the open, cross-platform object scripting language for enterprise networks and the Internet*. Dec. 1995. URL: http://wp.netscape.com/newsref/pr/newsrelease67.html (visited on 02/13/2008) (cit. on p. 29).

[41] Dave Raggett et al. *Reformulating HTML in XML*. W3C Recommendation. W3C, Dec. 1998. URL: http://www.w3.org/TR/1998/WD-html-in-xml-19981205/ (visited on 08/20/2015) (cit. on p. 31).

[42] Steven Pemberton et al. *XHTML™ 1.0: The Extensible HyperText Markup Language*. W3C Recommendation. W3C, Jan. 2000. URL: http://www.w3.org/TR/2000/REC-xhtml1-20000126/ (visited on 08/20/2015) (cit. on p. 31).

[43] T. Berners-Lee. *Linked Data*. Tech. rep. 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 09/17/2016) (cit. on p. 31).

[44] Ora Lassila and Ralph R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. W3C, Feb. 1999. URL: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ (visited on 08/18/2015) (cit. on pp. 31, 32).

[45] Dan Brickley and R. V. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ (visited on 08/18/2015) (cit. on p. 32).

[46] Deborah L. McGuinness and Frank van Harmelen. *OWL Web Ontology Language*. W3C Recommendation. W3C, Feb. 2004. URL: http://www.w3.org/TR/2004/REC-owl-features-20040210/ (visited on 08/18/2015) (cit. on p. 32).

[47] Dan Brickley and R. V. Guha. *JSON-LD 1.0: A JSON-based Serialization for Linked Data*. W3C Recommendation. W3C, Jan. 2014. URL: http://www.w3.org/TR/2014/REC-json-ld-20140116/ (visited on 08/19/2015) (cit. on p. 32).

[48] David Beckett et al. *RDF 1.1 Turtle*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-turtle-20140225/ (visited on 08/29/2015) (cit. on p. 32).

[49] David Beckett. *RDF 1.1 N-Triples*. W3C Recommendation. W3C, Feb. 2014. URL: http://www.w3.org/TR/2014/REC-n-triples-20140225/ (visited on 08/19/2015) (cit. on p. 32).

[50] Ben Adida et al. *RDFA in XHTML: Syntax and Processing*. W3C Recommendation. W3C, Oct. 2008. URL: http://www.w3.org/TR/2008/REC-rdfa-syntax-20081014/ (visited on 08/19/2015) (cit. on p. 32).

[51] Peter Schaffter. *What, exactly, is mom?* 2015. URL: http://www.schaffter.ca/mom/mom-01.html (visited on 09/16/2016) (cit. on p. 37).

[52] Donald Ervin Knuth. *Digital Typography*. The Center for the Study of Language and Information Publications, 1998. ISBN: 9780387982694 (cit. on p. 36).

[53] Albert Kapr. *Sto a jedna věta ke knižní úpravě*. Trans. by Antonín Rambousek. Lacerta, 1999. URL: http://www.sazba.cz/typoglosy/typo101.pdf (visited on 10/20/2015) (cit. on pp. 41, 46, 47).

[54] Robert Bringhurst. *the Elements of Typographic Style*. Point Roberts and Wash: Hartley & Marks, 1992. ISBN: 0881791105 (cit. on pp. 41, 42, 45–48).

[55] Matthew Butterick. *Butterick's Practical Typography: Line spacing*. URL: http://practicaltypography.com/line-spacing.html (visited on 11/02/2015) (cit. on p. 42).

[56] Vladimír Beran et al. *Aktualizovaný typografický manuál*. 6th ed. Kafka Design, 2014 (cit. on p. 45).

[57] CIE. ''Commission internationale de l'eclairage proceedings, 1931''. In: *Cambridge University Press Cambridge* (1932) (cit. on pp. 48, 49).

[58] Robert William Gainer Hunt. *The Reproduction of Colour*. 6th ed. John Wiley & Sons, 2004. ISBN: 0470024259 (cit. on p. 49).

[59] Wyszecki Günther and W. S. Styles. *Color science: Concepts and methods, quantitative data and formulae*. New York, 1982 (cit. on p. 49).

Acronyms

ACK The ACKnowledgement character
API Application Programming Interface
ASA The American Standard Association
ASCII The American Standard Code for Information Interchange
AT&T The American Telephone and Telegraph corporation
BEL The BELl character
BMP The Basic Multilingual Plane
BRE The Basic Regular Expressions
BS The BackSpace character
BSD The Berkeley Software Distribution. Also known as the Berkeley Unix
CA California
CAN The CANcel character
CERN The European Organization for Nuclear Research (*la Conseil Européen pour la Recherche Nucléaire*)
CIE The International Commission on Illumination (*Commission Internationale de l'Éclairage*)
CLDR The Common Locale Data Repository
CLI Command Line Interface
COBOL The COmmon Business-Oriented Language
CR The Carriage Return character
CSS The Cascading Style Sheets language
DC The Dublin Core
DC1 The Device Control character No. 1
DC2 The Device Control character No. 2
DC3 The Device Control character No. 3
DC4 The Device Control character No. 4
DEL The DELete character

DLE The Data Link Escape character
DPS Document Preparation System
DTD Document Type Declaration
DTP DeskTop Publishing
EBCDIC The Extended Binary Coded Decimal Interchange Code
ECMA The European Computer Manufacturers Association
EM The End of Medium
EMACS The Eventually Munches All Computer Storage editor
ENQ The ENQuiry character
EOT The End Of Transmission
ERE The Extended Regular Expressions
ESC The ESCape character
ETB The End of Transmission Block
ETX The End of TeXt
EUC The Extended Unix Code
FF The Form Feed character
FOAF Friend Or A Foe
FORTRAN The FORmula TRANslator
FS The File Separator
FSM The Free Software Movement
GML The General Markup Language
GNU GNU is Not Unix
GS The Group Separator
GUI Graphical User Interface
HT The Horizontal Tab
HTML The HyperText Markup Language
IBM The International Business Machines Corporation
ICC The International Color Consortium
IEC The International Electrotechnical Commission
IME Input Method Editor
IRI The Internationalized Resource Identifier
ISO The International Organization for Standardization
JIS The Japanese Industrial Standards encoding
JOE The Joe's Own Editor
JSON The JavaScript Object Notation
JSON-LD JSON for LD
JTC A Joint TC
LD Linked Data
LF The Line Feed
MA Massachusetts

MATHML The Mathematical Markup Language
NAK The Negative-AcKnowledgement character
NUL The NULl character
NY New York
OCR Optical Character Recognition
ODF The Open Document Format for office applications
OOXML The Office Open XML format
OWL The Web Ontology Language
PC The IBM Personal Computer
PDF The Portable Document Format
PICO The PIne COmposer
POSIX The Portable Operating System Interface
RDF The Resource Description Framework
RDFA RDF in attributes
RELAX NG The REgular LAnguage for XML New Generation
RFC A Request For Comments
RS The Record Separator
SC A SubCommittee
SGML The Standard General Markup Language
SI The Shift In character
SO The Shift Out character
SOH The Start of Heading
SPD Spectral Power Distribution
SR Sound Recognition
STX The Start of Text
SUB The SUBstitute character
SVG The Scalable Vector Graphics language
SVN SubVersioN
SYN The SYNchronous Idle character
TC A Technical Committee
TEI The Text Encoding Initiative
TRON The Real-time Operating system Nucleus
UCS The Universal multiple-octet coded Character Set
US The Unit Separator
USA The United States of America
UTF The UCS Transformation Format
VCS Version Control Systems
VI The Visual Interactive editor
VIM VI IMproved
VT The Vertical Tab

W3C The World Wide Web Consortium
WG A Working Group
WYSIWYG What You See Is What You Get
XHTML The eXtensible HyperText Markup Language
XML The eXtensible Markup Language

Index