Unicode Explained

Do you need to write a single software product or web site to target multiple platforms, languages, and character sets without re-engineering? There are hundreds of encoding systems for mapping characters to numbers, but Unicode promises a single mapping, which makes a single worldwide product solution possible. It’s no wonder that industry giants like Apple, Hewlett-Packard, IBM, and Microsoft have all adopted Unicode.

This comprehensive reference contains everything you need to understand Unicode. It takes you on a detailed tour through the complex character world. For starters, it explains how to identify and classify characters—from the common to the specialized. Then it shows you how to type these characters, interpret their properties, and process character data in a robust manner.

The first few chapters teach the basics of Unicode and character data. They provide a firm grasp of the terminology you need to make reference to various components, including:

- Character sets
- Fonts and encodings
- Glyphs and character repertoires

The middle section offers more detailed information about using Unicode and other character codes:

- Principles and methods behind defining character codes
- Some of the widely used codes
- Code conversion techniques
- Properties of characters
- Collation and sorting
- Line-breaking rules
- Unicode encodings

The final four chapters cover more advanced material, such as programming to support Unicode.

You simply can’t afford to be without the nuggets of valuable information detailed in *Unicode Explained*.

Jukka Korpela is a consultant who specializes in character codes, localization, orthography, usability, and accessibility. After graduating from the University of Helsinki, he taught these subjects at Helsinki University of Technology and other institutions. Later, he worked on localization and accessibility issues at TIEKE, the Finnish Information Society Development Center.
Table of Contents

Preface ................................................................. ix

Part I. Working with Characters

1. **Characters as Data** ................................................... 3
   - Introduction to Characters and Unicode 3
   - What’s in a Character? 6
   - Variation of Writing Systems 28
   - Glyphs and Fonts 30
   - Definitions of Character Repertoires 37
   - Numbering Characters 40
   - Encoding Characters as Octet Sequences 44
   - Working with Encodings 50
   - Working with Fonts 59
   - Summaries 67

2. **Writing Characters** .................................................... 71
   - Method Varieties 71
   - Keyboard Variation and Settings 75
   - Virtual Keyboards 79
   - Program Commands 83
   - Character Maps 91
   - Replacements on the Fly 96
   - Special Techniques 102
   - Escape Sequences 105
   - Specialized Editors 114
   - Exercise 115

3. **Character Sets and Encodings** ................................. 119
   - Good Old ASCII ........................................... 120
   - ISO 8859 Codes ........................................... 124
Windows Latin 1 and Other Windows Codes 127
Other 8-bit Codes 130
Unicode and UTF-8 138
Encodings for East Asian Language 143
Converters and Transcoding 146
Using Character Codes 148

Part II. A Systematic Look at Unicode

4. The Structure of Unicode ........................................... 157
   Design Principles 157
   Versions of Unicode 174
   Coding Space 175
   Unicode Terms 186
   Guide to the Unicode Standard 188
   Unicode and Fonts 198
   Criticism of Unicode 203
   Questions and Answers 212

5. Properties of Characters ........................................... 215
   Character Classification 216
   An Overview of Properties 219
   Compositions and Decompositions 230
   Normalization 244
   Case Properties 251
   Collation and Sorting 256
   Text Boundaries 264
   Directionality 265
   Line-Breaking Properties 272
   Unicode Conformance Requirements 290
   Effects on Choosing Characters 298

6. Unicode Encodings ................................................. 301
   Unicode Encodings in General 301
   UTF-32 and UCS-4 303
   UTF-16 and UCS-2 304
   UTF-8 306
   Byte Order 308
   Conversions Between Unicode Encodings 311
   Other Encodings 312
   Auto-Detecting the Encoding 326
   Choosing an Encoding 326
Part III. Advanced Unicode Topics

7. Characters and Languages ........................................... 333
   Writing Systems and IT .......................... 333
   Character Requirements of Languages ..... 350
   Transliteration and Transcription ........... 361
   Language Metadata ............................... 369
   Languages and Fonts ............................ 379

8. Character Usage .................................................. 383
   Basics of Character Usage ...................... 383
   ASCII (Basic Latin) .............................. 386
   Latin-1 Supplement (ISO 8859-1) ............ 406
   Other Latin Letters ............................. 414
   Other European Alphabetic Scripts .......... 414
   Diacritic Marks ................................... 416
   Letterlike Symbols .............................. 424
   General Punctuation ............................ 425
   Line Structure Control ........................ 438
   Mathematical and Technical Symbols ........ 442
   Other Blocks ..................................... 449

9. The Character Level and Above ................................ 459
   Levels of Text Representation and Processing 459
   Characters and Markup .......................... 480
   Media Types for Text ............................ 497

10. Characters in Internet Protocols ............................... 503
    Information About Encoding .................. 504
    Characters in MIME ............................ 509
    Content Negotiation and Multilingual Sites 532
    Characters in Protocol Headers .............. 545
    Characters in Domain Names and URLs ...... 549

11. Characters in Programming .................................... 553
    Characters in Computer Languages .......... 553
    Character and String Data ................. 564
    The Preparedness Principle ................. 585
    Character Input and Output ................. 592
    Processing Form Data ........................ 598
    Identifiers, Patterns, and Regular Expressions 602
    International Components for Unicode (ICU) 619
Characters often seem simple on the surface, but they are at the heart of a wide variety of data communications and data processing problems, including text processing, typesetting, styling text, text databases, and the transmission of textual information.

Computers were invented just for computing. For quite some time, they were so expensive that their use was limited to the most important numerical calculations that would have been impossible otherwise. Text was used mainly to add legends and headings to numeric output, often using a very limited character repertoire, maybe even lacking lowercase letters. As the cost of computing has dropped, computers have become extensively used for human communication in text format. Most people think of computers as communicators rather than calculators. People want to communicate in different languages, and we also use notation systems that may require rich repertoires of characters.

Unicode was developed to help make this both possible and smooth. Unicode was first defined in the early 1990s, but its use has progressed fairly slowly. Modern computers often use Unicode internally, but applications and users still tend to work with older character codes, which are often very limited. It has been rather complicated to work with Unicode in text processing, for example. At long last, however, these problems are becoming easier to solve. Information technology is becoming really multinational, supporting different languages, writing systems, and conventions. IT products need to be at least potentially suitable for use in different cultural environments, or “localizable.” Unicode itself is just part of the technical basis for all this, but it is an indispensable part.

The technological basis of using Unicode, though still imperfect, is much better than most people’s capabilities for making use of it. Even computer professionals often don’t know how to work with large repertoires of characters. The bottleneck is lack of a basic knowledge and skills, not a lack of hardware or software.

The concept of a character is one of the most difficult basic concepts in information technology, yet fundamental to text processing, databases, the Web, XML-based mark-up, internationalization, and other areas. People who encounter Unicode when studying such topics often remain confused and frustrated. They mostly find material that assumes that the reader already knows what Unicode is. It might be even worse: it is very
easy to find incorrect or seriously confusing information about Unicode and characters, even in new books. People find themselves in a maze of twisty little passages of characters, fonts, encodings, and related concepts.

This book guides you through the Unicode and character world. It explains how to identify and classify characters—whether common, uncommon, or exotic—and to type them, to use their properties, and to process character data in a robust manner. It helps you to live in a world with several character encodings.

**Audience**

Readers of this book are expected to be familiar with computers and how computers work, broadly speaking. They are not expected to know computer programming, though many readers will use the contents in system design and programming.

This book is intended for people with different backgrounds and needs, including:

- An end user of multilingual or specialized text-related applications. For example, anyone who works with texts containing mathematical or special symbols, or uses a multilingual database. These readers should probably explore Chapters 1 through 3 first, practice with that content, and then read Chapters 7 and 8.

- An IT professional who needs to understand Unicode and work with it. The need might arise from text data conversion tasks, from creating internationalized software or web sites, or from system design or programming in an environment that uses Unicode.

- An IT teacher who needs a better understanding of character code issues, both to understand the subject area better and to disseminate correct information. There is rather little about character codes in curricula, and this is largely a chicken-and-egg problem: there are no good textbooks, and teachers themselves don’t know the topic well enough. The first three chapters of the book could provide the foundation for a course, optionally coupled with other chapters relevant to a particular curriculum.

- An IT student, hobbyist, or professional who keeps hearing about Unicode and needs to work with technologies that use Unicode, such as XML.

**Assumptions and Approach**

Previous knowledge about character codes is not assumed. If you already know about them, you may need to change your mental model a bit.

This book starts at the ordinary computer user’s level. Thus, it unavoidably contains explanations that look trivial to some readers. However, these discussions might help in explaining things to others when needed. The book also contains practical instructions on actually working with “special” characters, and an IT professional might find
this irrelevant. However, studying such issues and practicing with them will help a lot in creating a background for more technical work with the infrastructures of character usage.

In explaining practical ways of doing things, this book often uses Microsoft Windows and Microsoft Office programs as examples. This is because so many people use such software and need to know how to use Unicode in them. Moreover, even if you personally prefer other software, odds are good that you need to work with Windows and Office at times. Information on using Unicode in some other environments can be found in the following:

- Markus Kuhn: “UTF-8 and Unicode FAQ for Unix/Linux,” which is available at http://www.cl.cam.ac.uk/~mgk25/unicode.html
- Tom Gewecke: “Unleash Your Multilingual Mac,” which is available at http://hometown.aol.com/tg3907/mlingos9.html

After the first three chapters, this book gets more technical, and many of the issues discussed are abstract and even formal. Therefore, understanding most of the material in the initial chapters is essential for the rest. To most people, it is very difficult to read about abstract things if you lack a concrete background that lets you map the abstract concepts and rules to specific practice.

This book explores Unicode processing generally, but cannot go into great detail on all parts of the Unicode character space. For much more information on ideographic characters and processing of East Asian languages, see Ken Lunde’s CJKV Information Processing (O’Reilly).

Except for the last chapter (Chapter 11), this book does not assume that the reader knows about computer programming. However, some references to programming are made throughout the book.

**Contents of This Book**

The book has three parts:

**Part I**

Chapters 1 through 3 provide a self-contained tutorial presentation of Unicode and character data. It is aimed at anyone who has a basic understanding of computing, and introduces characters in information technology, with some historical background. Although much of this part is well-known to many IT professionals, it provides a consistent terminology that could give professionals (and especially teachers) a model for talking to laymen about characters.

**Part II**

Chapters 4 through 6 give detailed information about using Unicode and other character codes. These chapters are specifically aimed at computer science students and teachers, information technology professionals, and people involved in lin-
guistic data processing and databases containing string data. Together with the first part, this covers what every IT professional should know about characters. It explains the principles and methods of defining character codes, describes some of the widely used codes, presents code conversion techniques, and takes a detailed look at Unicode. This includes properties and classification of characters, collation and sorting, line breaking rules, and Unicode encodings.

Part III

Chapters 7 through 11 discuss relatively independent topics, to be read according to each reader’s specific needs. They are topics that are important and even crucial to many, but not necessary to all. For example, if you need to author or administer multilingual web sites, you should read the section on characters in HTML and XHTML. To be honest, I would suggest that most people need to read it at least twice. Character code problems are intrinsically difficult, and very widely misunderstood. It takes time to digest the concepts and principles before you can really start working with the algorithms and tools.

The chapters can be characterized as follows:

Chapter 1, *Characters as Data*

This chapter describes, at a general level but exemplified by simple and typical cases, how computers represent and process characters. It defines fundamental concepts like character set, code position, encoding, glyph, and font. At this point, Unicode is the only character set discussed, to avoid confusion. To make the discussion more concrete and motivating, some features of writing systems are described. The historical development of character codes is presented to the extent that is necessary for understanding why even apparently simple characters, such as dashes and é, still cause problems. The use of different encodings is illustrated by examples of viewing email messages and web pages, using commands to select the encoding if needed. The basic methods for finding, installing, and selecting fonts are described.

Chapter 2, *Writing Characters*

This is a practical presentation of some common methods of entering characters, including keyboard variation, special keys, changing keyboard settings, virtual keyboards, character maps, automatic “correction” of character sequences, program commands, and different escape notations. It is largely a collection of recipes, useful, for example, to people who work daily with texts containing “difficult” characters. For this reason, some quick reference tables for very commonly needed characters are presented. However, it is also relevant to IT specialists who need to understand the possible input methods when designing applications and systems. The examples used are mostly from MS Windows and MS Office environments but various alternatives (such as “Unicode editors”) are also discussed. HTML and XML character reference and entity reference techniques are presented as well. The chapter ends with an appendix for writing some specialized texts using some of the techniques presented.
Chapter 3, *Character Sets and Encodings*

This chapter describes some very widely used character codes and encodings, mainly ASCII, ISO-8859-1 and other ISO-8859 standards, Windows Latin 1 and relatives, and UTF-8. (However, the semantics of characters are described in Chapter 8.) Some less common encodings such as DOS code pages are described in order to give some basics for working with legacy data and legacy systems. A few widely used multibyte encodings for East Asian languages are briefly described, too. The section describes how conversions between the encodings can be performed, either with the functions of commonly used programs or separate converters. It also discusses practical feasibility of the character sets in different contexts, such as email, Internet discussion forums, and document interchange. MIME is presented to the extent needed for dealing with the charset issue.

Chapter 4, *The Structure of Unicode*

An in-depth presentation of the fundamentals of Unicode, including design principles, coding space, and special terminology. The nature of Unicode as an umbrella standard based on a large number of older standards is explained, as well as its relationship to ISO 10646. The unification principle as well as criticism of it is described.

Chapter 5, *Properties of Characters*

This chapter describes the various properties defined for characters in the Unicode standard and their relationship with some programming concepts. This is, in part, a companion to the much more formal definitions in the standard itself. In particular, compatibility, decompositions, collation, sorting, directionality, and line-breaking properties as well as Unicode normalization forms are described.

Chapter 6, *Unicode Encodings*

This chapter describes UTF-8 and other Unicode encodings in detail, including the algorithmic descriptions and the practical considerations on choosing an encoding.

Chapter 7, *Characters and Languages*

The chapter describes some IT-related requirements of different languages and writing systems, such as how to deal with right-to-left writing. This includes conversions between writing systems (transliteration or transcription). The interaction between encoding, language, and font settings is described. Moreover, language codes, language metadata, and language markup are described, illustrated with XML examples.

Chapter 8, *Character Usage*

This chapter consists of sections devoted to different character blocks and collections that are practically important especially in the Western world. The first section is more generic and discusses the relationship of character standards, orthography, and typography. (Even in purely English-language text, typographically correct punctuation requires characters beyond ASCII.) The chapter contains detailed information about the semantics and usage of individual characters, although the level of detail depends greatly on the importance of the character. All
the major blocks are briefly characterized to give an overview, but the emphasis is on ASCII, different Latin supplements, general punctuation, and mathematical and technical symbols.

Chapter 9, *The Character Level and Above*

Characters form but one “protocol level,” above which there are higher levels such as markup level, record structure level, and application level. This chapter provides guidelines for the coding of information at different levels when there is choice, such as using markup versus character difference (largely still an open problem despite the efforts of the W3C and the Unicode Consortium). This is particularly important for processing of legacy data and for avoiding overly fine distinctions at the character level. The chapter ends with a section on media types for text and the difference between plain text, other subtypes of text, and application types such as text-processing formats.

Chapter 10, *Characters in Internet Protocols*

This chapter describes how character encoding information is transmitted using Internet protocols, including MIME and HTTP, and how content negotiation works on the Web (for the purposes of negotiating on character encoding). This constitutes a basis for a presentation of some fundamentals of multilingual web authoring at the technical level. Moreover, the use of characters in the protocols themselves, such as Internet message headers and URLs, is described, with focus on the partial shift from pure ASCII to Unicode. In particular, the technical basis of Internationalized Domain Names and Internationalized URLs is described.

Chapter 11, *Characters in Programming*

This chapter presents a number of ways to represent character and string data in different programming languages, such as FORTRAN, C, C#, Perl, ECMAScript, and Java™, as well as other computer languages such as XML and CSS. It emphasizes both the differences and similarities, which are illustrated with sample programs to perform simple manipulation of string data. The chapter is especially intended for people who teach programming but also for people who study or practice programming in an environment where character data is essential. Programs that cannot distinguish, for example, between an empty string, a space character, the NUL character, and the digit zero will have large problems in a Unicode environment. The chapter also examines requirements for modern processing of character data, including the principle of being prepared to handle a large character repertoire and that of separating internal encoding from input and output encodings. The International Components for Unicode (ICU) activity and its results are described. The chapter also contains a section on Common Locale Data Repository (CLDR) and its future use in disciplined programming. This largely goes beyond the character concept but is motivated by the use of Unicode in CLDR and by the organizational connection with the Unicode Consortium.
Appendix, *Tables for Writing Characters*

The Appendix provides some commonly needed information useful for entering characters. It includes tables of key sequences, as well as a mapping chart from the Symbol font to Unicode.

**Self-Assessment Test**

To estimate your progress in knowledge about Unicode, you can perform the following self-assessment test. Read the following statements and comment on each of them with one of the following alternatives (using whatever symbols you find convenient, such as those in parentheses): “I do not understand what the statement says” (??), “I know what it says but I do not know whether it is true” (?), “true” (+), and false (–). Moreover, for any “true” or “false” answer, consider what you would present as an argument in a discussion in which someone says you’re wrong.

At any point in reading the book, and especially when you think you have learned enough, reread the statements and perform the test again. You might regard the following as a spoiler, so it has been written backward so that you can hopefully ignore it at this point if you like. It reveals what the test is about: .elpoep ot siht nialpxe ot deen thgim uoy dna ,gnorw era yeht yhw wonk ot laitnesse si ti ecnis ,hguoht ,siht gniwonk htiw deifsitas eb ton dluohs uoY .eslaf lla era yeht tub ,skoob ecnerefer ni neve edam ylnommoc era stnemetats ehT

1. Unicode is a 16-bit character code.
2. Unicode contains all the characters used in the languages of the world.
3. Unicode is meant to replace all the other character codes.
4. Unicode cannot be used in real applications now; it is just a future plan.
5. Using Unicode, the size of a text file gets doubled.
6. We don’t need Unicode if we write only in English.
7. Unicode consists of 256 code pages.

**Conventions Used in This Book**

The following typographical conventions are used in this book:

*Italic*

Indicates new terms, URLs, email addresses, filenames, and file extensions.

*Constant width*

Indicates computer code in a broad sense. This includes commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties (does *not* include Unicode “properties”), parameters,
values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, and the output from commands.

**Constant width bold**
Shows commands or other text that should be typed literally by the user.

**Constant width italic**
Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

The following special notations are used in this book to refer to characters:

“x”
Refers to character x by showing it within double quotation marks. For clarity, characters that might be confused with other characters in the text—i.e., letters a–z, A–Z, and some common punctuation, such as hyphens (-), commas (,), and periods (.)—are enclosed in quotation marks.

U+nnnn
Refers to a character (or a code point) by its Unicode number. The number nnnn is written in hexadecimal notation, usually in four digits using leading zeros if needed.

Web sites and pages are mentioned in this book to help the reader locate online information that might be useful. Normally both the address (URL) and the name (title, heading) of a page are mentioned. Some addresses are relatively complicated, but you can probably locate the pages easily by using your favorite search engine to find a page by its name, typically by typing it inside quotation marks. This may also help if the page cannot be found by its address; it may have moved elsewhere, so the name may work.

**Using Code Examples**

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you’re reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O’Reilly books does require permission. Answering a question by citing this book and quoting example
code does not require permission. Incorporating a significant amount of example code from this book into your product’s documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: “Unicode Explained by Jukka K. Korpela. Copyright 2006 O’Reilly Media, Inc., 0-596-10121-X.”

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

**Safari® Enabled**

When you see a Safari® Enabled icon on the cover of your favorite technology book, that means the book is available online through the O’Reilly Network Safari Bookshelf.

Safari offers a solution that’s better than e-books. It’s a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at [http://safari.oreilly.com](http://safari.oreilly.com).

**How to Contact Us**

Please address comments and questions concerning this book to the publisher:

O’Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:


To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O’Reilly Network, see our web site at:

[http://www.oreilly.com](http://www.oreilly.com)
Acknowledgments

The presentation of problems, solutions, and ideas owes much to people with whom I have been in contact in character-related matters through years, such as (roughly in chronological order by their influence) Timo Kiravuo, Alan J. Flavell, Arjun Ray, Roman Czyborra, Bob Bemer, and Erkki I. Kolehmainen.

The reviewers, Andreas Prilop, John Cowan, and Jori Mäntysalo gave a very substantial amount of valuable input, both on content and on presentation. Simon St.Laurent has had an active and supportive role through the entire process as an editor.
This part describes the fundamentals of representing character data in computers, including Unicode and other important character codes. It also discusses several practical ways of writing Unicode characters.
Computers were originally built to process numbers. Over the last few decades, they’ve become increasingly better at handling text as well, but the transition from human scribbling and beautiful typography to bits and bytes has been complicated. Going from a paper document to a computerized representation of that document means learning about how the computer handles text, and requires learning about characters, character codes, fonts, and encodings. Unicode provides a set of solutions for some of these problems, while retaining presentation flexibility for making text look as we feel it should.

Introduction to Characters and Unicode

Computer programs use two basic data types in most of their processing: characters and numbers. These basic types are combined in various ways to create strings, arrays, records, and other data structures. (Inside the computer, characters are numbers, but the ways that these numbers are handled is very different from numbers meant for calculation.)

Early computers were largely oriented toward numerical computation. However, characters were used early on in administrative data processing, where names, addresses, and other data needed to be stored and printed as strings. Text processing on computers became more common much later, when computers had become so affordable that they replaced typewriters. At present, most text documents are produced and processed using computers.

Originally, character data on computers had limited types and uses. For economic and technical reasons, the repertoire of characters was very small, not much more than the letters, digits, and basic punctuation used in normal English. This constitutes but a tiny fraction of the different characters used in the world’s writing systems—about 100 characters out of literally myriads (tens of thousands) of characters. Thus, there was a growing need for a possibility of presenting and handling a large character repertoire on computers; Unicode is the fundamental answer to that.
**Why Unicode?**

Since you are reading this book, I assume you already have sufficient motivation to learn about Unicode. Nevertheless, a short presentation follows that explains the benefits of Unicode.

Computers internally work on numbers. This means that characters need to be coded as numbers. A typical arrangement is to use numbers from 0 to 255, because that range fits into a basic unit of data storage and transfer, called a *(8-bit) byte or octet.*

When you define how those numbers correspond to characters, you define a *character code.* There are quite a number of character codes defined and used in the world. Most of them have the same assignments for numbers 0 to 127, used for characters that appear in English as well as in many other languages: the letters a–z plus their uppercase equivalents, the digits 0–9, and a few punctuation marks. Many of the code numbers in this so-called ASCII set of characters are used for various technical purposes.

For French texts, for example, you need additional characters such as accented letters (é, ô, etc.). These can be provided by using code numbers in the range 128–255 in addition to the ASCII range, and this gives room for letters used in most other Western European languages as well. Thus, you can use a single character code, called Latin 1, even for a text containing a mixture of English, French, Spanish, and German, because these languages all use the Latin characters with relatively few additions.

However, you quickly run out of numbers if you try to cover too many languages within 256 characters. For this reason, different character codes were developed. For example, Latin 1 is for Western European languages, Latin 2 for several languages spoken in Central and Eastern Europe, and additional character codes exist for Greek, Cyrillic, Arabic, etc. When only one language is used, you can usually pick up a suitable character code and use it. In fact, someone probably did that for you when designing the particular computer system (including software) that you use. You may have used a particular character code for years without knowing anything about it.

Character codes that use only the code numbers from 0 to 255 are called *8-bit codes,* since such code numbers can be represented using 8 bits.

Things change when you need to combine languages in one document and the languages are fundamentally different in their use of characters. In an English-German or French-Spanish glossary, for example, you can use Latin 1. In English-Greek data, you can use one of the character codes developed for Greek, since these codes contain the ASCII characters. But what about French-Greek? That’s not possible the same way, since the character codes discussed above do not support such a combination. A code either has Latin accented letters in the “upper half” (the range of 128–255), or it has Greek letters (α, β, γ, etc.) there. It would be impractical, and often impossible, to define 256-character codes for all the possible language combinations.

As you probably know, the number of characters needed for Chinese and Japanese is very large. They just would not fit into a set with only 256 characters. Therefore, dif-
ferent strategies are used. For example, 2 bytes (octets) instead of one might be used for one character. This would give 65,536 possible numbers for a character. On the other hand, the character codes developed for the needs of East Asian languages do not contain all the characters used in the world.

The solution to such problems, and many other problems in the world of growing information exchange, is the introduction of a character code that gives every character of every language a unique number. This number does not depend on the language used in the text, the font used to display the character, the software, the operating system, or the device. It is universal and kept unchanged. The range of possible numbers is set sufficiently high to cover all the current and future needs of all languages.

The solution is called Unicode, and it gives anyone the opportunity to say, “I want this character displayed and the number is...” and have herself understood by all systems that support Unicode. This does not always guarantee a success in displaying the character, due to lack of a suitable font, but such technical problems are manageable.

Much widely used software, including Microsoft Windows, Mac OS X, and Linux, has supported Unicode for years. However, to use Unicode, all the relevant components must be “Unicode enabled.” For example, although Windows “knows Unicode,” an application program used on a Windows system might not. Moreover, the display or printing of characters often fails since fonts (software for drawing characters) are still incomplete in covering the set of Unicode characters. This is changing as more complete fonts become available and as programs become more clever in their ability to use characters from different fonts.

Unicode Can Be Easy

Unicode is both very easy and very complicated. The fundamental principles are simple and natural, as the explanation above hopefully illustrated. The actual typing and viewing of Unicode characters can also be easy, when modern tools are used. As we get to complicated issues like sorting Unicode strings or controlling line breaking, you will find some challenges. But this book starts from simple principles and usage.

For example, an average PC running the Windows XP system has a universal tool for typing any Unicode character, assuming that it is contained in some font installed on the system. The tool is called the Character Map, or CharMap for short. Figure 1-1 shows the user interface of this program. The program can be launched from the Start menu, although you may need to look for it among “System tools” or something like that. You can select a collection of characters from a menu, and then click on a character to select it. The selected characters can be copied onto the clipboard with a single click, and you can then paste them (e.g., with Ctrl-V) where you like.

There are many other similar tools, often with advanced character search features. There are also ways to configure your keyboard on the fly so that keys and key combinations produce characters that you use frequently.
What’s in a Character?

We use characters daily: we type them, and we read them on screen or on paper. We use text-processing programs routinely, much like people used to use typewriters, pens, or other writing tools. How could characters create problems?

Why Do We Need to Know About Characters?

If English is your native language, you are accustomed to using a small set of characters, consisting of the letters A–Z and a–z, digits 0–9, and a few punctuation characters. Most novels, newspaper articles, and memos contain no other characters. Since you seem to be able to type these characters directly on a keyboard, why should you learn more about characters and get confused? To be honest, character issues are confusing.

Suppose you use a computer only to write and edit texts in English, perhaps as a secretary or a technical editor. You will have reasons to know about characters:

Figure 1-1. Character Map, part of Windows XP, lets you type any Unicode character
• Computer technology has caused a decline in typography, and you can make a positive impression by using correct punctuation instead of typewriter-style punctuation. If you use a text-processing program, it probably takes care of using “smart” quotation marks instead of "straight" quotes, but you need to learn how to produce dashes—like this—and how to prevent bad line breaks.

• Normal English texts may contain special characters occasionally. Someone may spell Caesar as Cæsar, or use a word like fiancé, rôle, or garçon the French way, or use the per mille sign ‰ or the euro sign €. Michael Everson writes: “Despite unfounded but widespread belief to the contrary (based doubtless on the prevalence of ASCII), diacritics (usually French ones) are often found in naturalized English words. Examples are: à la carte, abbé, Ægean, archaeology, belovèd, café, décor, détente, éclair, façade, fête, naïve, naïvety (but cf. non-naturalized naïveté), noël, œsophagus, résumé, vicuña” (http://www.evertype.com/alphabets/english.pdf). You may regard some of these spellings as foreign or obsolete, but people may still use them in English. There are often good reasons to change the spelling to something simpler, but not knowing how to produce the characters is not a good reason.

• Your text may contain foreign names with some strange characters. Although it is common to simplify the spelling, you can stand out positively by doing things correctly. Suppose that someone’s surname is Hämäläinen and she works in an important international position. She is probably accustomed to seeing her name written as Hamalainen or Haemaelaeinen. But wouldn’t she be delighted if someone were polite enough and competent enough to spell her name right, just for a change? However, she might not like it if someone tried to do so and failed, producing Hmlinen or H{m{l{inen.

• You might even be asked to include quotations in a foreign language. You might even need to work with a document in a foreign language, because someone has to do that and this is your day for being that someone. In that case, you may need to use foreign punctuation as well and to find a way to enter foreign characters efficiently, in addition to just knowing a universal clumsy way of entering any character.

• Texts increasingly contain technical and scientific special notations. Even casual memos and messages may need to mention μm (micrometer) or to use the almost equals sign ≈ or the male sign ♂. In scientific or technical texts, mathematical formulas are often quite crucial and need to be exactly right, down to the choice of each special symbol. The world is getting more technical and symbolic. Even nontechnical texts like bridge columns contain special symbols, such as ♠.

In multilingual applications, characters and their codes are a major issue. Even a web site with two or more languages or a bilingual dictionary can be regarded as multilingual applications, and they create the problem of representing the characters of both or all languages. For example, the computer user using English and people using Russian on computers probably work with their own tools, settings, and conventions, but if you need to create
a document that is bilingual in French and Russian, you need to make sure you can work with both Latin letters with diacritic marks and Cyrillic letters. In effect, you would need to use Unicode, one way or another.

If you are a computer professional, you need to be prepared to handle data-processing problems that may involve characters of any kind. Someday someone will ask you to work with a system for processing data in a strange language or with strange symbols in it, perhaps even in a writing system where text runs right to left. It will be very difficult if you have no background in working with such issues. Most people need quite some time to digest character problems and techniques. You may find that, with something you thought you knew for years, you have completely misunderstood some basics.

Even if you process only “normal” text, character code standards and specifications are more important than they used to be. Modularity of software requires that you isolate character-level processing from other levels. You should not test for a character variable’s value being equal to 32 to test whether it is a space character. Often, even a more sensible test, against the character constant '\', is suboptimal, and using a built-in function like `isspace` is better, since it takes care of other space-like characters as well. Tools developed for such operations are increasingly based on general specification in character standards, especially the Unicode standard. They are supposed to define, in a systematic and all-compassing way, the fundamental properties of characters, like being space-like, or being a letter, or allowing a line break before or after a character. To use such definitions and software modules that implement them, you don’t need to know every detail, but you need to know the principles and the ways to get at the details when needed.

In addition, if you design or develop programs, databases, or systems, you will find that it is extremely difficult to adapt them to processing different character sets, if they were not designed to work that way. If the software is full of code that relies on using 1 byte (octet, 8-bit entity) for one character, it may need an almost complete rewrite if it needs to be modified to process Chinese text as well.

**Characters as Units of Text**

A character is a basic (or “atomic”) unit of written text. A piece of text is a sequence of characters, also called a string. This does not necessarily mean that text is always displayed so that its characters appear linearly one after another, although this is what happens for English text, if we ignore the issue of division into lines. In other writing systems, consecutive characters may be combined into one glyph in complex ways. However, the text is still logically a sequence of characters.

**Characters as abstractions**

To store, process, and transfer data in digital form, we need an abstract concept of a character. It would not be feasible to store the specific appearance of each written character. Instead, we store information that tells which character it is, independent of
the specific visual shape it has. If we wish to affect the way in which our characters are
displayed and printed, we use special formatting commands or other tools.

The abstract concept of character is essential in Unicode, in all digital processing of
character data, and even in writing itself. The meaning of a piece of text does not change
if you change its font, the specific design of its characters. To put it a bit differently,
the style and taste—and even the effect—of text might change, but we have an intuitive
understanding of something invariant behind such variation. For example, “A,” “א,”
“A,” and “א” are instances of the same character. Since you know the Latin alphabet,
you should have no difficulty with this. You might find it more difficult to know
whether א and א are instances of the same Hebrew character, but people who speak
Hebrew are able to recognize that.

Different attempts have been made to describe what characters are. They have even
been compared to Platonic forms. The point is that there is so much negative in the
concept: it is largely defined by saying what a character is not. In a sense, we extract
properties and concrete features, until there’s very little left—something that could be
called the idea of a particular character. Dan Connolly has written in his classical treatise
“Character Set’ Considered Harmful”: “Note that by the term character, we do not
mean a glyph, a name, a phoneme, nor a bit combination. A character is simply an
atomic unit of communication. It is typically a symbol whose various representations
are understood to mean the same thing by a community of people.”

This raises the question of what to do if different people recognize things differently.
In some languages, “v” and “w” have been treated as typographic variants of a single
character; other languages treat them as completely distinct letters. In such situations,
Unicode normally defines separate characters.

To clarify the abstract nature of characters, a Unicode character, or a character defined
by some other standard:

- Normally has no particular stylistic appearance but may vary between broad limits,
  as long as the designs can be recognized as the same character
- Is essentially black and white, though a character as a whole could be colored with
  any other two colors (making, for example, the ♥ character appear in red), using
  methods external to character standards
- Has an official name (as described later) but no fixed name across languages, and
  not necessarily any commonly known name in a particular language
- Has no fixed pronunciation, except for some specifically phonetic characters;
  however, there are of course correspondences between letters and sounds, even
  across languages that use the same basic writing system
- May have very specific usage as a special symbol (e.g., © is just a copyright symbol)
  or a broad range of different uses (e.g., / can be a separator of a kind, a mathematical
  operator, or something else)
Variation of appearance or different characters?

Problems arise when the concept of an abstract character has to be applied to concrete situations. We know what the letter “A” is, but is it the same as the lowercase letter “a”? That is, is the difference between them just variation in appearance, the same way as the letter “A” in the Times font differs from the letter “A” in the Arial font? In fact, the lowercase letters are a medieval invention, created by people who wrote text by hand and needed forms that are more convenient for that.

We could have defined “A” and “a” as just visual variants of the same abstract character, but we didn’t. Quite early in the history of computers, this decision was made. It has far-reaching implications. If you wish to process input data so that upper- and lowercase letters are equivalent, to make things easier to people who type the data, you need to do something special to take care of that.

To take things a bit further, consider the Latin letter “A” and its relationship to the corresponding Cyrillic letter and the corresponding Greek letter, capital alpha. All three letters look the same in most fonts, and they share a common origin. Yet they belong to different alphabets: the Latin alphabet A, B, C, D..., which we use in English and many other languages, the Cyrillic alphabet А, Б, В, Г..., which is used in Russian and many Eastern European languages, and the Greek alphabet Α, Β, Γ, Δ... (alpha, beta, gamma, delta...).

It would have been possible to identify the Latin “A” and its Cyrillic and Greek counterparts. However, it was decided to keep them separate. Generally, Unicode (and character standards in general) do not unify characters across writing system boundaries. We might take this just as a fact of life and live with it. But we might also look at its reasonableness. Consider the operation of converting text from upper- to lowercase. The Latin letter “A” should become “a,” whereas the Greek letter alpha “Α” should become α. It would be impossible to do this automatically if it were impossible to tell, from the internal digital representation, whether the original data contains the Latin “A” or the Greek “Α.”

Writing systems were invented by people, and characters are creations of mankind, not nature. Thus, the identity of abstract characters is in a sense just a decision made by some people. However, it is usually an informed decision.

Variation in shape turned into a character difference

In many cases, stylistic variation in drawing or printing a character has been “frozen” so that a variant obtains a specific shape and meaning. The ancient Romans used the letter “V” both as a consonant and as vowel. Later, it appeared in different variants, such as a rounded one, like our “U.” People started using the original version and different curved variants in different contexts. As such usage became systematic, consistent, and common, the letter “U” was born.
Therefore, we now have the independent characters “V” and “U.” They are, in turn, written with stylistic variation, though now the general idea is that the variation should not obscure the difference between these two characters. Yet, you might still see “V” used for “U” for stylistic reasons, especially to imitate ancient inscriptions (SENATVS POPVLVSQVE ROMANVS).

The letters “U” and “V” have later given birth to new characters that have originally been formed as their typographic variants, as well as the letter “W,” originally a digraph (VV). Special forms of this letter have been recognized as separate characters, such as the modifier letter small w, “w”. The story goes on. In different areas that need new symbols, characters are created as variants or modifications of old characters. This seems to suit the human mind better than the invention of new character shapes from scratch.

**Characters and “abstract characters”**

The Unicode standard defines different meanings for the term *character*. The first one is: “The smallest component of written language that has semantic value; refers to the abstract meaning and/or shape, rather than a specific shape (see also *glyph*), though in code tables some form of visual representation is essential for the reader’s understanding.” The second meaning is that “character” is a synonym for “abstract character,” which is defined as “a unit of information used for the organization, control, or representation of textual data.”

Thus, the difference seems to be that an abstract character may have a control purpose only. Control purposes include line breaks, for example. In more common terminology, “character” in Unicode often means a printable (graphic) character, whereas “abstract character” means what is commonly called just “character,” which includes printable and control characters.

On the other hand, the Unicode standard also uses the expression “abstract character” to refer to a symbol that may be perceived by users as a character (“user character”), although it cannot be represented as a single Unicode character (also known as encoded character or coded character). In particular, a symbol with special marks (diacritic marks) on it, such as ó, cannot always be represented as one character in Unicode but may be a sequence of two or more characters.

The expression “semantic value” is somewhat misleading in this context. A character such as a letter can hardly be described as having a meaning (semantic value) in itself. It would be better to say that a character has a *recognized identity* and it may be sometimes used as meaningful in itself (as a symbol or as a one-letter word) but more often as a component of a string that has a meaning. Moreover, the “smallest component” part is somewhat vague. A character such as ú (letter u with an acute accent), which belongs to Unicode, can often be regarded as consisting of smaller components: a letter and a diacritic (acute accent). In fact, in Unicode, the character ú may be regarded *either* as a character on its own or as a combination of two successive characters, letter “u” and a combining acute accent.
The intuitive concept of character varies by language and cultural background. If you know the letter ä mainly from J. R. R. Tolkien’s books, you might regard it just as letter “a” with a special mark that indicates that it is to be pronounced separately. You might even regard the two dots just as optional decoration, as in “naïve” if spelled in the French way. If your native language were Finnish, you would certainly treat ä as a completely separate character, and you would have learned at school that it has its own position in alphabetic order (a, b, c,...x, y, z, â, ä, ö). Similarly, in Swedish, the words “här” (“here”), “har” (“has”), and “hår” (“hair”) must be kept clearly separate. To a German, ä is different from “a,” but it is treated as primarily equivalent to “a” in alphabetic order and is in a sense a variant of “a” (“an Umlaut”).

Unicode, aiming at universality, generally recognizes written forms as separate characters, if at least one language or commonly used notation system makes a difference. Thus, “a” and ä are treated as distinct. If you wish to handle them as equivalent, you need to program code that treats them that way.

Characters and other units of text

Although a character is a natural “atom” of text in data processing, it does not always correspond to people’s intuitive idea of the basic constituents of text. Looking at text in English, we might occasionally ask ourselves whether the ligature fi is two characters or one. In other writing systems, similar questions arise more often. Unicode takes a liberal approach to identifying a complex character in many cases. You can represent fi as one character or (more often) as two characters, “f” and “i.” As mentioned above, similar principles apply to letters with diacritic marks.

People who speak languages with many diacritic marks or ligatures may regard a symbol like fi or ú as a single character, even though they are often coded as sequences of characters. In some cases, it would not even be possible to code the symbol as a single character in Unicode, since Unicode does not contain all the combinations and ligatures that can be formed.

Moreover, although characters might be written separately, as in “ch,” their combination might be understood as a single entity by some people. In English, “ch” denotes a particular sound and has thus some identity of its own. Some other languages treat the combination as an inseparable unit even in alphabetic order: in a dictionary, words would appear in an order like car, czar, char. Such treatment has become less common, though, since it is somewhat more difficult to implement in automated processing. Unicode treats “ch” as two characters but recognizes that it might constitute a unit in ordering.

Partly for such reasons, the ordering of characters is rather complex. Unicode does not prescribe a single ordering of characters and strings. Rather, it defines a basic (default) ordering that can be used as basis for defining language-dependent and even application-specific orderings.
Characters Versus Images

Characters are normally represented in graphic form, as something that can be called an image. However, there is a fundamental difference between an image and a character. An image can be a particular rendering of a character, much like a spoken word is a particular presentation of an element of a language. Moreover, most images are not renderings of characters at all.

Character code standards mostly identify a symbol as a character only if it is actually used in texts—e.g., in books, magazines, newspapers, and electronic documents. Characters that are normally used only in product labels and other specialized contexts are often borderline cases. However, they are often identified as characters if they are used in conjunction with symbols that are undeniably characters.

A typical example is the estimated symbol ℮, a stylized variant of the letter “e.” It is not used in normal texts, but only in European packaging to claim conformance to certain standards in specifying a quantity. However, it is identified as a character, partly because it is used in packages in relation to text characters—e.g., in “℮ 200 g” (indicating that the mass of the product is 200 grams, within tolerances defined in specific regulation).

On the other hand, logos and identifying symbols are not treated as characters, even though they might be accompanied by texts. By its nature, a logo consists of a name or abbreviation in a particular graphic style. Hence, it would be unnatural to encode it as a character or sequence of characters, although we might use a string of characters as a replacement for a logo (e.g., when a document containing a logo needs to be converted to plain text and the logo conveys essential information).

Similarly, most of the various political, ideological, or religious symbols are treated as graphic symbols that are not characters. They are not normally used in texts. Their shape may vary, but not as part of font variation. However, for various reasons, some graphic symbols have been defined as characters in some character codes, contrary to these principles. Unicode therefore contains them as characters, so that existing texts using such characters can be encoded.

Generally, a graphic symbol is encoded as a character in Unicode, if there is need for exchanging it in digital form in plain text. Decisions on this are sometimes difficult and may be affected by tradition.

The distinction between a character and an image is often a practical decision to be made by the author or editor of a document. In many cases, you have a choice between a character and an image. For example, suppose that you are designing a user interface for a document, program, or web page and you need graphic symbols for “Next” and “Previous.” It may often be best to use characters; let us assume that you want to use
arrows pointing to the left and to the right. Beware that even at this fairly abstract level, the decision is not culturally neutral: it implies left-to-right writing direction.

In Unicode, there is a largish block of arrow characters. Among them, a few like ← and → are widely available in commonly used fonts. However, they are not very prominent graphically, even if shown in bold, in large font, and in color. Their graphic design is character-like, not iconic. Some other characters in the Arrows block of Unicode look more solid, but they are not as common in fonts. For buttons or links, specially designed images may thus work better. On the other hand, in running texts, the arrow characters often work well. If you wish to make references to other entries in an encyclopedia by using arrows, then “→foobar” works better than a word preceded by a distinctive graphic.

Generally, when deciding between the use of characters and the use of an image for presenting a graphic symbol, the following items should be considered:

- Are there some Unicode characters that could be used, and are they suitable both by their defined semantics and by their typical graphic appearance?
- Is it possible that the document will be rendered so that images are not displayed? If yes, is it possible to specify a textual alternative to the image (such as the alt attribute in HTML markup)?
- How safely would the character work, given all the possible problems with encodings, fonts, etc.?
- Is it acceptable, and perhaps desirable, that the symbol changes size, shape, or color when text font size, face, or color is changed?
- Is it possible that the data will be processed as a character string—e.g., stored in a database or used in a search string?

For example, suppose we write about music and wish to refer to F-sharp and B-flat using the conventional musical symbols: F♯, B♭. The Unicode approach would use the special characters: music sharp sign ♯ and music flat sign ♭. However, these characters, although part of Unicode since Version 1.1, are poorly supported in fonts. Even though you could find them in some fonts at your disposal, their appearance might not fit into your typographic design. You might end up using the number sign # and the letter “b” as replacements. In web authoring, for example, you might decide that although B&amp;#x266d; would be technically quite correct (using a so-called character reference to include the flat sign), it is safer to create a small image, say flat.gif, and embed it with markup like B<img src="flat.gif" alt="♭">. This means that the flat symbol remains in constant size if the text size is changed, but this is usually tolerable.

Sometimes character-looking symbols are not characters. Microsoft Word by default changes the three-character sequence “—>” into a kind of arrow symbol (à). However, this arrow is different from any Unicode character: it is just a glyph in the Wingdings font. It is therefore something between a character and an image; as so many compromises, it combines the drawbacks of the alternatives.
Processing of Characters

The previous discussion mentioned that characters can be processed and used in many ways that are not possible (or practical), if information is represented as images, sounds, or in another nontext format. This includes:

• Searching for occurrences of a word or other fragment of text, using either a simple search string or a text pattern
• Performing automatic replacements, such as substituting a string for another in all occurrences
• Indexing the data for efficiency of searching and for creating an alphabetic index or concordance (list of occurrences of words)
• Sorting text data—e.g., for presentation in alphabetic order
• Copying text from an application or data format to another, often via a clipboard
• Modifying text as in a text editor or text-processing application, by deleting, inserting, and replacing characters
• Selecting parts of text by user actions, such as painting or keyboard commands
• Recognizing constructs like words, syllables, morphemes (components of a word with a meaning), and sentences
• Computing statistics on the use of characters, words, phrases, etc.
• Spelling and grammar checks
• Automatic or computer-aided translation
• Presenting texts in audible form, via speech synthesis, which is more natural these days than you might expect from many science fiction films

Even the display of characters on screen or paper involves processing:

• Choice of font, which can be a complex process
• Application of bolding, italics, and other features, if requested
• Selection of contextual forms for characters
• Recognition of character sequences that should or could be rendered using ligatures or other special methods
• Formation of characters with diacritic marks, often requiring complex algorithms
• Adjusting spacing between characters and words, perhaps for justification of lines
• Breaking text into lines, perhaps using hyphenation

In particular, suppose that some document exists on paper only, or as a scanned image only. The above lists of possibilities can be consulted when estimating whether the text should be converted into text format. The conversion may require quite a bit of work, including the identification of special characters occurring in the documents.
About the Author

Jukka Korpela is a consultant who specializes in character codes, localization, orthography, usability, and accessibility. After graduating from Helsinki University of Technology, he taught these subjects in the university's Computer Science department and worked on localization and accessibility issues at TIEKE before becoming a full-time author and consultant. His previous books on CSS and XHTML were published in Finland by Docendo press.

Colophon

The animal on the cover of Unicode Explained is a long-tailed glossy starling (Lamprotornis caudatus). So named because of its lustrous plumage, this bird is indigenous to tropical parts of Africa, stretching from Senegal to Sudan. Glossy starlings are a common sight in that part of the world, and the bird's harsh, noisy call makes it difficult to miss.

Long-tailed glossy starlings spend most of their time in areas of open woodland, feeding on a mix of fruit and insects. Nests are normally built in a hole in a tree trunk, and a female starling will usually lay two to four eggs at a time. The feathers on young glossy-tailed starlings are dull brown and are not nearly as bright as adult feathers. Starlings grow to be about 21 inches, 13 of which is their long, striking tail.

The cover image is from the Dover Pictorial Archive. The cover font is Adobe ITC Garamond. The text font is Times New Roman; the heading font is Adobe Myriad Condensed; the table font is Arial Unicode MS; and the code font is LucasFont's TheSans Mono Condensed. Special characters that don't exist in the main body font are primarily in Arial Unicode MS. However, some special characters required particular fonts: Everson Mono Unicode, Microsoft Times New Roman, Microsoft Wingdings, Georgia, Symbol, Tahoma, and Verdana.