The Unicode® Standard

Version 16.0 – Core Specification

The Unicode Consortium

Starting with this version, the Unicode Consortium has changed the way the Unicode Standard is produced. The interactive HTML version is authoritative. There is also an archival PDF version. See *Section C.1.16, Unicode 16.0*, for details.

Contents

Preface

Why Unicode?

The Unicode Standard and its associated specifications provide programmers with a single universal character encoding, extensive descriptions, and a vast amount of data about how characters function. The specifications and data describe how to form words and break lines; how to sort text in different languages; how to format numbers, dates, times, and other elements appropriate to different languages; how to display languages whose written form flows from right to left, such as Arabic and Hebrew, or whose written form splits, combines, and reorders, such as languages of South Asia. These specifications include descriptions of how to deal with security concerns regarding the many "look-alike" characters from alphabets around the world. Without the properties and algorithms in the Unicode Standard and its associated specifications, interoperability between different implementations would be impossible, and much of the vast breadth of the world's languages would lie outside the reach of modern software.

Organization of This Standard

This core specification, together with the Unicode code charts, the Unicode Character Database, and the Unicode Standard Annexes, defines the Unicode Standard. The core specification contains the general principles, requirements for conformance, and guidelines for implementers. The character code charts and names are available online.

Concepts, Architecture, Conformance, and Guidelines. The first five chapters introduce the Unicode Standard and provide the fundamental information needed to produce a conforming implementation. Basic text processing, working with combining marks, encoding forms, and normalization are all described. A special chapter on implementation guidelines answers many common questions that arise when implementing Unicode.

Chapter 1 introduces the standard's basic concepts, design basis, and coverage and discusses basic text handling requirements.

Chapter 2 sets forth the fundamental principles underlying the Unicode Standard and covers specific topics such as text processes, overall character properties, and the use of combining marks.

Chapter 3 constitutes the formal statement of conformance. This chapter also presents the normative algorithms for several processes, including normalization, Korean syllable boundary determination, and default casing.

Chapter 4 describes character properties in detail, both normative (required) and informative. Additional character property information appears in Unicode Standard Annex #44, "Unicode Character Database."

Chapter 5 discusses implementation issues, including compression, strategies for dealing with unknown and unsupported characters, and transcoding to other standards.

Character Block Descriptions. *Chapters 6 through 23* contain the character block descriptions that provide basic information about each script or group of symbols and may discuss specific characters or pertinent layout information. Some of this information is required to produce conformant implementations of these scripts and other collections of characters.

Code Charts. *Chapter 24* describes the conventions used in the code charts and the list of character names. The code charts contain the normative character encoding assignments, and the names list contains normative information, as well as useful cross references and informational notes.

Appendices. The appendices contain additional information.

Appendix A documents the notational conventions used by the standard.

Appendix B provides information about Unicode publications and links to other important Unicode resources.

Appendix C details the relationship between the Unicode Standard and ISO/IEC 10646.

Appendix D lists version history.

Appendix E describes the history of Han unification in the Unicode Standard.

Appendix F provides additional documentation for characters encoded in the CJK Strokes block (U+31C0..U+31EF).

Online Information. A glossary of Unicode terms, the Unicode Character Name Index, and the list of references for the Unicode Standard are located at:

https://www.unicode.org/glossary/

https://www.unicode.org/charts/charindex.html

https://www.unicode.org/references/

The Unicode Character Database

The Unicode Character Database (UCD) is a collection of data files containing character code points, character names, and character property data. It is described more fully in *Section 4.1, Unicode Character Database* and in Unicode Standard Annex #44, "Unicode Character Database." All versions, including the most up-to-date version of the Unicode Character Database, are found at:

https://www.unicode.org/ucd/

Information on versioning and on all versions of the Unicode Standard can be found at:

https://www.unicode.org/versions/

Unicode Code Charts

The Unicode code charts contain the character encoding assignments and the names list. The archival, reference set of versioned 16.0 code charts may be found at:

https://www.unicode.org/charts/PDF/Unicode-16.0/

For easy lookup of characters, see the current code charts:

https://www.unicode.org/charts/

An interactive radical-stroke index to CJK ideographs is located at:

https://www.unicode.org/charts/unihanrsindex.html

Unicode Standard Annexes

The Unicode Standard Annexes form an integral part of the Unicode Standard. Conformance to a version of the Unicode Standard includes conformance to its Unicode Standard Annexes. All versions, including the most up-to-date versions of all Unicode Standard Annexes, are available at:

https://www.unicode.org/reports/index.html#annexes

The following is the list of Unicode Standard Annexes:

Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," describes specifications for the positioning of characters in text containing characters flowing from right to left, such as Arabic or Hebrew.

Unicode Standard Annex #11, "East Asian Width," presents the specification of an informative property for Unicode characters that is useful when interoperating with East Asian legacy character sets.

Unicode Standard Annex #14, "Unicode Line Breaking Algorithm," presents the specification of line breaking properties for Unicode characters.

Unicode Standard Annex #15, "Unicode Normalization Forms," describes Unicode normalization and provides examples and implementation strategies for it.

Unicode Standard Annex #24, "Unicode Script Property," describes two related Unicode code point properties. Both properties share the use of Script property values. The Script property itself assigns single script values to all Unicode code points, identifying a primary script association, where possible. The Script_Extensions property assigns sets of Script property values, providing more detail for cases where characters are commonly used with multiple scripts.

Unicode Standard Annex #29, "Unicode Text Segmentation," describes algorithms for determining default boundaries between certain significant text elements: grapheme clusters ("user-perceived characters"), words, and sentences.

Unicode Standard Annex #31, "Unicode Identifiers and Syntax," describes specifications for recommended defaults for the use of Unicode in the definitions of identifiers and in pattern-based syntax.

Unicode Standard Annex #34, "Unicode Named Character Sequences," defines the concept of a Unicode named character sequence.

Unicode Standard Annex #38, "Unicode Han Database (Unihan)," describes the organization and content of the Unihan Database.

Unicode Standard Annex #41, "Common References for Unicode Standard Annexes," contains the listing of references shared by other Unicode Standard Annexes.

Unicode Standard Annex #42, "Unicode Character Database in XML," describes an XML representation of the Unicode Character Database.

Unicode Standard Annex #44, "Unicode Character Database," provides the core documentation for the Unicode Character Database (UCD). It describes the layout and organization of the Unicode Character Database and how the UCD specifies the formal definition of Unicode character properties.

Unicode Standard Annex #45, "U-Source Ideographs," describes U-source ideographs as used by the Ideographic Research Group (IRG) in its CJK ideograph unification work.

Unicode Standard Annex #50, "Unicode Vertical Text Layout," describes the Unicode character property, Vertical_Orientation, which can serve as a stable default orientation for characters for reliable document interchange.

Unicode Standard Annex #53, "Unicode Arabic Mark Rendering," specifies an algorithm that can be utilized during rendering for determining correct display of Arabic combining mark sequences.

Unicode Standard Annex #57, "Unicode Egyptian Hieroglyph Database (Unikemet)," describes the organization and content of the Unikemet Database.

Unicode Technical Standards and Unicode Technical Reports

Unicode Technical Reports and Unicode Technical Standards are separate publications and do not form part of the Unicode Standard. However, several Unicode Technical Standards are versioned synchronously with the Unicode Standard and have newly published versions:

Unicode Technical Standard #10, "Unicode Collation Algorithm," details how to compare two Unicode strings while remaining conformant to the requirements of the Unicode Standard. It includes the Default Unicode Collation Element Table (DUCET) and conformance tests.

Unicode Technical Standard #39, "Unicode Security Mechanisms," specifies mechanisms that can be used to detect possible security problems involving Unicode characters. It includes data tables for confusable characters.

Unicode Technical Standard #46, "Unicode IDNA Compatibility Processing," discusses compatibility between IDNA 2003, IDNA 2008, and current browser practice for domain names. It provides a comprehensive mapping to support current user expectations for casing and other variants of domain names.

Unicode Technical Standard #51, "Unicode Emoji," defines the structure of Unicode emoji characters and sequences, and provides data to support that structure, such as which characters are considered to be emoji, and which emoji should be displayed by default with a text style versus an emoji style. It also provides design guidelines for improving the interoperability of emoji characters across platforms and implementations.

All versions of all Unicode Technical Reports and Unicode Technical Standards are available at:

https://www.unicode.org/reports/

Updates and Errata

Reports of errors in the Unicode Standard, including the Unicode Character Database and the Unicode Standard Annexes, may be reported using the reporting form:

https://corp.unicode.org/reporting/error.html

A list of known errata is maintained at:

https://www.unicode.org/errata/

Any currently listed errata will be fixed in subsequent versions of the standard.

Acknowledgements

The Unicode Standard is the result of the dedication and contributions of numerous people over many years. We would like to acknowledge the individuals whose contributions were central to the design, authorship, and review of this standard. A complete listing of acknowledgements can be found at:

https://www.unicode.org/acknowledgements/standard.html

There is also a page dedicated specifically to acknowledgement of contributors of the many fonts used in production of the Unicode Standard:

https://www.unicode.org/charts/fonts.html

Current editorial contributors can be found at:

https://www.unicode.org/consortium/edcom.html

About This Publication

The core specification is built as a static website with the Astro framework and Svelte components. The archival PDF version is generated with WeasyPrint. Example glyphs are shaped with harfbuzzjs. The text is mainly set in STIX Two Text. Most of the figures were created with Adobe Illustrator.

The Unicode code charts were produced with Unibook chart formatting software supplied by ASMUS, Inc.

Chapter 1

Introduction

The Unicode Standard is the universal character encoding standard for written characters and text. It defines a consistent way of encoding multilingual text that enables the exchange of text data internationally and creates the foundation for global software. As the default encoding of HTML and XML, the Unicode Standard provides the underpinning for the World Wide Web and the global business environments of today. Required in new Internet protocols and implemented in all modern operating systems and computer languages such as Java and C#, Unicode is the basis of software that must function all around the world.

With Unicode, the information technology industry has replaced proliferating character sets with data stability, global interoperability and data interchange, simplified software, and reduced development costs.

While taking the ASCII character set as its starting point, the Unicode Standard goes far beyond ASCII's limited ability to encode only the upper- and lowercase letters A through Z. It provides the capacity to encode all characters used for the written languages of the world—more than 1 million characters can be encoded. No escape sequence or control code is required to specify any character in any language. The Unicode character encoding treats alphabetic characters, ideographic characters, and symbols equivalently, which means they can be used in any mixture and with equal facility (see *Figure 1-1*).

Figure 1-1. Wide ASCII

ASCII/8859-1 Text		Unicode Text	
A	0100 0001	A	0000 0000 0100 0001
S	0101 0011	S	0000 0000 0101 0011
C	0100 0011	C	0000 0000 0100 0011
I	0100 1001	I	0000 0000 0100 1001
I	0100 1001	I	0000 0000 0100 1001
/	0010 1111		0000 0000 0010 0000
8	0011 1000	天	0101 1001 0010 1001
8	0011 1000	地	0101 0111 0011 0000
5	0011 0101		0000 0000 0010 0000
9	0011 1001	س	0000 0110 0011 0011
-	0010 1101	ل	0000 0110 0100 0100
1	0011 0001	ا	0000 0110 0010 0111
	0010 0000	م	0000 0110 0100 0101
t	0111 0100		0000 0000 0010 0000
e	0110 0101	α	0000 0011 1011 0001
x	0111 1000	∉	0010 0010 0111 0000
t	0111 0100	γ	0000 0011 1011 0011

The Unicode Standard specifies a numeric value (code point) and a name for each of its characters. In this respect, it is similar to other character encoding standards from ASCII onward. In addition to character codes and names, other information is crucial to ensure legible text: a character's case, directionality, and alphabetic properties must be well defined. The Unicode Standard defines these and other semantic values, and it includes application data such as case mapping tables and character property tables as part of the Unicode Character Database. Character properties define a character's identity and behavior; they ensure consistency in the processing and interchange of Unicode data. See *Section 4.1, Unicode Character Database*.

Unicode characters are represented in one of three encoding forms: a 32-bit form (UTF-32), a 16-bit form (UTF-16), and an 8-bit form (UTF-8). The 8-bit, byte-oriented form, UTF-8, has been designed for ease of use with existing ASCII-based systems.

The Unicode Standard is code-for-code identical with International Standard ISO/IEC 10646. Any implementation that is conformant to Unicode is therefore conformant to ISO/IEC 10646.

The Unicode Standard contains 1,114,112 code points, most of which are available for encoding of characters. The majority of the common characters used in the major

languages of the world are encoded in the first 65,536 code points, also known as the Basic Multilingual Plane (BMP). The overall capacity for more than 1 million characters is more than sufficient for all known character encoding requirements, including full coverage of all minority and historic scripts of the world.

1.1 Coverage

The Unicode Standard, Version 16.0, contains over 150,000 characters from the world's scripts. These characters are more than sufficient not only for modern communication for the world's languages, but also to represent the classical forms of many languages. The standard includes the European alphabetic scripts, Middle Eastern right-to-left scripts, and scripts of Asia and Africa. Many archaic and historic scripts are encoded. The Han script includes 97,680 unified ideographic characters defined by national, international, and industry standards of China, Japan, Korea, Taiwan, Vietnam, and Singapore. In addition, the Unicode Standard contains many important symbol sets, including currency symbols, punctuation marks, mathematical symbols, technical symbols, geometric shapes, dingbats, and emoji. For overall character and code range information, see *Chapter 2, General Structure*.

Note, however, that the Unicode Standard does not encode idiosyncratic, personal, novel, or private-use characters, nor does it encode logos or graphics. Graphologies unrelated to text, such as dance notations, are likewise outside the scope of the Unicode Standard. Font variants are explicitly not encoded. The Unicode Standard reserves 6,400 code points in the BMP for private use, which may be used to assign codes to characters not included in the repertoire of the Unicode Standard. Another 131,068 private-use code points are available outside the BMP, should 6,400 prove insufficient for particular applications.

1.1.1 Standards Coverage

The Unicode Standard is a superset of all characters in widespread use today. It contains the characters from major international and national standards as well as prominent industry character sets. For example, Unicode incorporates the ISO/IEC 6937 and ISO/IEC 8859 families of standards, the SGML standard ISO/IEC 8879, and bibliographic standards such as ISO 5426. Important national standards contained within Unicode include ANSI Z39.64, KS X 1001, JIS X 0208, JIS X 0212, JIS X 0213, GB 2312, GB 18030, HKSCS, and CNS 11643. Industry code pages and character sets from Adobe, Apple, Fujitsu, Hewlett-Packard, IBM, Lotus, Microsoft, NEC, and Xerox are fully represented as well.

The Unicode Standard is fully conformant with the International Standard ISO/IEC 10646:2020, *Information Technology—Universal Coded Character Set (UCS),* known as the Universal Character Set (UCS). For more information, see *Appendix C, Relationship to ISO/IEC 10646*.

1.1.2 New Characters

The Unicode Standard continues to respond to new and changing industry demands by encoding important new characters. As the universal character encoding, the

Unicode Standard also responds to scholarly needs. To preserve world cultural heritage, important archaic scripts are encoded as consensus about the encoding is developed.

1.2 Design Goals

The Unicode Standard began with a simple goal: to unify the many hundreds of conflicting ways to encode characters, replacing them with a single, universal standard. The pre-existing legacy character encodings were both inconsistent and incomplete—two encodings could use the same codes for two different characters and use different codes for the same characters, while none of the encodings handled any more than a small fraction of the world's languages. Whenever textual data was converted between different programs or platforms, there was a substantial risk of corruption. Programs often were written only to support particular encodings, making development of international versions expensive. As a result, developing countries were particularly hard-hit, as it was not economically feasible to adapt specific versions of programs for smaller markets. Technical fields such as mathematics were also disadvantaged, because they were forced to use special fonts to represent arbitrary characters, often leading to garbled content.

The designers of the Unicode Standard envisioned a uniform method of character identification that would be more efficient and flexible than previous encoding systems. The new system would satisfy the needs of technical and multilingual computing and would encode a broad range of characters for all purposes, including worldwide publication.

The Unicode Standard was designed to be:

- *Universal.* The repertoire must be large enough to encompass all characters that are likely to be used in general text interchange, including those in major international, national, and industry character sets.

- *Efficient.* Plain text is simple to parse: software does not have to maintain state or look for special escape sequences, and character synchronization from any point in a character stream is quick and unambiguous. A fixed character code allows for efficient sorting, searching, display, and editing of text.

- *Unambiguous.* Any given Unicode code point always represents the same character.

Figure 1-2 demonstrates some of these features, contrasting the Unicode encoding with mixtures of single-byte character sets with escape sequences to shift the meanings of bytes in the ISO/IEC 2022 framework using multiple character encoding standards.

Figure 1-2. Unicode Compared to the 2022 Framework
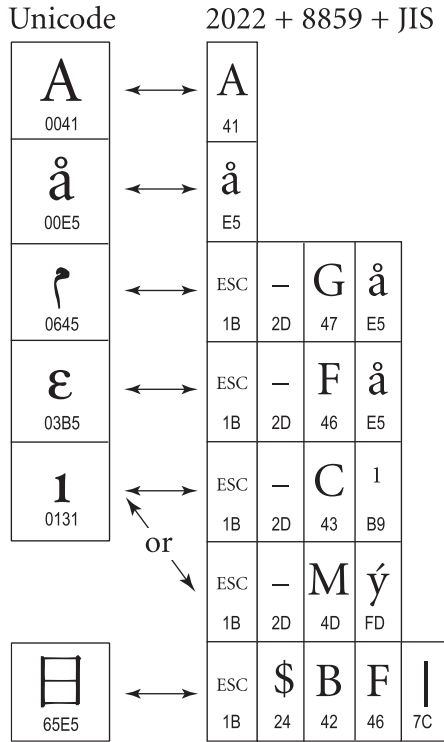
1.3 Text Handling

The assignment of characters is only a small fraction of what the Unicode Standard and its associated specifications provide. The specifications give programmers extensive descriptions and a vast amount of data about the handling of text, including how to:

- divide words and break lines
- sort text in different languages
- format numbers, dates, times, and other elements appropriate to different locales
- display text for languages whose written form flows from right to left, such as Arabic or Hebrew
- display text in which the written form splits, combines, and reorders, such as for the languages of South Asia
- deal with security concerns regarding the many look-alike characters from writing systems around the world

Without the properties, algorithms, and other specifications in the Unicode Standard and its associated specifications, interoperability between different implementations would be impossible. With the Unicode Standard as the foundation of text representation, all of the text on the Web can be stored, searched, and matched with the same program code.

1.3.1 Characters and Glyphs

The difference between identifying a character and rendering it on screen or paper is crucial to understanding the Unicode Standard's role in text processing. The character identified by a Unicode code point is an abstract entity, such as "LATIN CAPITAL LETTER A" or "BENGALI DIGIT FIVE". The mark made on screen or paper, called a glyph, is a visual representation of the character.

The Unicode Standard does not define glyph images. That is, the standard defines how characters are interpreted, not how glyphs are rendered. Ultimately, the software or hardware rendering engine of a computer is responsible for the appearance of the characters on the screen. The Unicode Standard does not specify the precise shape, size, or orientation of on-screen characters.

1.3.2 Text Elements

The successful encoding, processing, and interpretation of text requires appropriate definition of useful elements of text and the basic rules for interpreting text. The defi-

nition of text elements often changes depending on the process that handles the text. For example, when searching for a particular word or character written with the Latin script, one often wishes to ignore differences of case. However, correct spelling within a document requires case sensitivity.

The Unicode Standard does not define what is and is not a text element in different processes; instead, it defines elements called *encoded characters*. An encoded character is represented by a number from 0 to 10FFFF_{16}, called a code point. A text element, in turn, is represented by a sequence of one or more encoded characters.

Chapter 2

General Structure

This chapter describes the fundamental principles governing the design of the Unicode Standard and presents an informal overview of its main features. The chapter starts by placing the Unicode Standard in an architectural context by discussing the nature of text representation and text processing and its bearing on character encoding decisions. Next, the Unicode Design Principles are introduced—ten basic principles that convey the essence of the standard. The Unicode Design Principles serve as a tutorial framework for understanding the Unicode Standard.

The chapter then moves on to the Unicode character encoding model, introducing the concepts of character, code point, and encoding forms, and diagramming the relationships between them. This provides an explanation of the encoding forms UTF-8, UTF-16, and UTF-32 and some general guidelines regarding the circumstances under which one form would be preferable to another.

The sections on Unicode allocation then describe the overall structure of the Unicode codespace, showing a summary of the code charts and the locations of blocks of characters associated with different scripts or sets of symbols.

Next, the chapter discusses the issue of writing direction and introduces several special types of characters important for understanding the Unicode Standard. In particular, the use of combining characters, the byte order mark, and other special characters is explored in some detail.

The section on equivalent sequences and normalization describes the issue of multiple equivalent representations of Unicode text and explains how text can be transformed to use a unique and preferred representation for each character sequence.

Finally, there is an informal statement of the conformance requirements for the Unicode Standard. This informal statement, with a number of easy-to-understand examples, gives a general sense of what conformance to the Unicode Standard means. The rigorous, formal definition of conformance is given in the subsequent <u>Chapter 3, Conformance</u>.

2.1 Architectural Context

A character code standard such as the Unicode Standard enables the implementation of useful processes operating on textual data. The interesting end products are not the character codes but rather the text processes, because these directly serve the needs of a system's users. Character codes are like nuts and bolts—minor, but essential and ubiquitous components used in many different ways in the construction of computer software systems. No single design of a character set can be optimal for all uses, so the architecture of the Unicode Standard strikes a balance among several competing requirements.

2.1.1 Basic Text Processes

Most computer systems provide low-level functionality for a small number of basic text processes from which more sophisticated text-processing capabilities are built. The following text processes are supported by most computer systems to some degree:

- Rendering characters visible (including ligatures, contextual forms, and so on)
- Breaking lines while rendering (including hyphenation)
- Modifying appearance, such as point size, kerning, underlining, slant, and weight (light, demi, bold, and so on)
- Determining units such as "word" and "sentence"
- Interacting with users in processes such as selecting and highlighting text
- Accepting keyboard input and editing stored text through insertion and deletion
- Comparing text in operations such as in searching or determining the sort order of two strings
- Analyzing text content in operations such as spell-checking, hyphenation, and parsing morphology (that is, determining word roots, stems, and affixes)
- Treating text as bulk data for operations such as compressing and decompressing, truncating, transmitting, and receiving

2.1.2 Text Elements, Characters, and Text Processes

One of the more profound challenges in designing a character encoding stems from the fact that there is no universal set of fundamental units of text. Instead, the division of text into *text elements* necessarily varies by language and text process.

For example, in the pre-1996 German orthography, the letter combination "ck" was a text element for the process of hyphenation (where it appeared as "k-k"), but not for

the process of sorting. In Spanish, the combination "ll" may be a text element for the traditional process of sorting (where it is sorted between "l" and "m"), but not for the process of rendering. In English, the letters "A" and "a" are usually distinct text elements for the process of rendering, but generally not distinct for the process of searching text. The text elements in a given language depend upon the specific text process; a text element for spell-checking may have different boundaries from a text element for sorting purposes. For example, in the phrase "the quick brown fox," the sequence "fox" is a text element for the purpose of spell-checking.

In contrast, a character encoding standard provides a single set of fundamental units of encoding, to which it uniquely assigns numerical code points. These units, called *assigned characters*, are the smallest interpretable units of stored text. Text elements are then represented by a sequence of one or more characters.

Figure 2-1 illustrates the relationship between several different types of text elements and the characters used to represent those text elements.

Figure 2-1. Text Elements and Characters
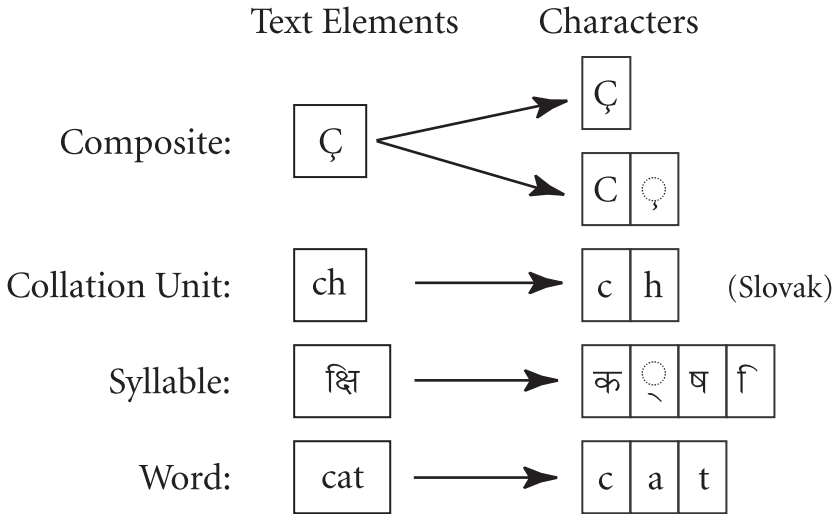
The design of the character encoding must provide precisely the set of characters that allows programmers to design applications capable of implementing a variety of text processes in the desired languages. Therefore, the text elements encountered in most text processes are represented as sequences of character codes. See Unicode Standard Annex #29, "Unicode Text Segmentation," for detailed information on how to segment character strings into common types of text elements. Certain text elements correspond to what users perceive as single characters. These are called *grapheme clusters*.

2.1.3 Text Processes and Encoding

In the case of English text using an encoding scheme such as ASCII, the relationships between the encoding and the basic text processes built on it are seemingly straightforward: characters are generally rendered visible one by one in distinct rectangles from left to right in linear order. Thus one character code inside the computer corresponds to one logical character in a process such as simple English rendering.

When designing an international and multilingual text encoding such as the Unicode Standard, the relationship between the encoding and implementation of basic text processes must be considered explicitly, for several reasons:

- Many assumptions about character rendering that hold true for the English alphabet fail for other writing systems. Characters in these other writing systems are not necessarily rendered visible one by one in rectangles from left to right. In many cases, character positioning is quite complex and does not proceed in a linear fashion. See *Section 9.2, Arabic*, and *Section 12.1, Devanagari*, for detailed examples of this situation.

- It is not always obvious that one set of text characters is an optimal encoding for a given language. For example, two approaches exist for the encoding of accented characters commonly used in French or Swedish: ISO/IEC 8859 defines letters such as "ä" and "ö" as individual characters, whereas ISO 5426 represents them by composition with diacritics instead. In the Swedish language, both are considered distinct letters of the alphabet, following the letter "z". In French, the diaeresis on a vowel merely marks it as being pronounced in isolation. In practice, both approaches can be used to implement either language.

- No encoding can support all basic text processes equally well. As a result, some trade-offs are necessary. For example, following common practice, Unicode defines separate codes for uppercase and lowercase letters. This choice causes some text processes, such as rendering, to be carried out more easily, but other processes, such as comparison, to become more difficult. A different encoding design for English, such as case-shift control codes, would have the opposite effect. In designing a new encoding scheme for complex scripts, such trade-offs must be evaluated and decisions made explicitly.

For these reasons, design of the Unicode Standard is not specific to the design of particular basic text-processing algorithms. Instead, it provides an encoding that can be used with a wide variety of algorithms. In particular, sorting and string comparison algorithms *cannot* assume that the assignment of Unicode character code numbers provides an alphabetical ordering for lexicographic string comparison. Culturally expected sorting orders require arbitrarily complex sorting algorithms. The expected sort sequence for the same characters differs across languages; thus, in general, no single acceptable lexicographic ordering exists. See Unicode Technical

Standard #10, "Unicode Collation Algorithm," for the standard default mechanism for comparing Unicode strings.

Text processes supporting many languages are often more complex than they are for English. The character encoding design of the Unicode Standard strives to minimize this additional complexity, enabling modern computer systems to interchange, render, and manipulate text in a user's own script and language—and possibly in other languages as well.

Character Identity. Whenever Unicode makes statements about the default layout behavior of characters, it is done to ensure that users and implementers face no ambiguities as to which characters or character sequences to use for a given purpose. For bidirectional writing systems, this includes the specification of the sequence in which characters are to be encoded so as to correspond to a specific reading order when displayed. See *Section 2.10, Writing Direction*.

The actual layout in an implementation may differ in detail. A mathematical layout system, for example, will have many additional, domain-specific rules for layout, but a well-designed system leaves no ambiguities as to which character codes are to be used for a given aspect of the mathematical expression being encoded.

The purpose of defining Unicode default layout behavior is not to enforce a single and specific aesthetic layout for each script, but rather to encourage uniformity in encoding. In that way implementers of layout systems can rely on the fact that users would have chosen a particular character sequence for a given purpose, and users can rely on the fact that implementers will create a layout for a particular character sequence that matches the intent of the user to within the capabilities or technical limitations of the implementation.

In other words, two users who are familiar with the standard and who are presented with the same text ideally will choose the same sequence of character codes to encode the text. In actual practice there are many limitations, so this goal cannot always be realized.

2.2 Unicode Design Principles

The design of the Unicode Standard reflects the 10 fundamental principles stated in *Table 2-1*. Not all of these principles can be satisfied simultaneously. The design strikes a balance between maintaining consistency for the sake of simplicity and efficiency and maintaining compatibility for interchange with existing standards.

Table 2-1. The 10 Unicode Design Principles

Principle	Statement
Universality	The Unicode Standard provides a single, universal repertoire.
Efficiency	Unicode text is simple to parse and process.
Characters, not glyphs	The Unicode Standard encodes characters, not glyphs.
Semantics	Characters have well-defined semantics.
Plain text	Unicode characters represent plain text.
Logical order	The default for memory representation is logical order.
Unification	The Unicode Standard unifies duplicate characters within scripts across languages.
Dynamic composition	Accented forms can be dynamically composed.
Stability	Characters, once assigned, cannot be reassigned and key properties are immutable.
Convertibility	Accurate convertibility is guaranteed between the Unicode Standard and other widely accepted standards.

2.2.1 Universality

The Unicode Standard encodes a single, very large set of characters, encompassing all the characters needed for worldwide use. This single repertoire is intended to be universal in coverage, containing all the characters for textual representation in all modern writing systems, in most historic writing systems, and for symbols used in plain text.

The Unicode Standard is designed to meet the needs of diverse user communities within each language, serving business, educational, liturgical and scientific users, and covering the needs of both modern and historical texts.

Despite its aim of universality, the Unicode Standard considers the following to be outside its scope: writing systems for which insufficient information is available to enable reliable encoding of characters, writing systems that have not become standardized through use, and writing systems that are nontextual in nature.

Because the universal repertoire is known and well defined in the standard, it is possible to specify a rich set of character semantics. By relying on those character semantics, implementations can provide detailed support for complex operations on text in a portable way. See "Semantics" later in this section.

2.2.2 Efficiency

The Unicode Standard is designed to make efficient implementation possible. There are no escape characters or shift states in the Unicode character encoding model. Each character code has the same status as any other character code; all codes are equally accessible.

The standard Unicode encoding forms (UTF-8, UTF-16, and UTF-32) are self-synchronizing and non-overlapping. This makes randomly accessing and searching inside streams of characters efficient.

By convention, characters of a script are grouped together as far as is practical. Not only is this practice convenient for looking up characters in the code charts, but it makes implementations more compact and compression methods more efficient. The common punctuation characters are shared.

Format characters are given specific and unambiguous functions in the Unicode Standard. This design simplifies the support of subsets. To keep implementations simple and efficient, stateful controls and format characters are avoided wherever possible.

2.2.3 Characters, Not Glyphs

The Unicode Standard draws a distinction between *characters* and *glyphs*. Characters are the abstract representations of the smallest components of written language that have semantic value. They represent primarily, but not exclusively, the letters, punctuation, and other signs that constitute natural language text and technical notation. The letters used in natural language text are grouped into scripts—sets of letters that are used together in writing languages. Letters in different scripts, even when they correspond either semantically or graphically, are represented in Unicode by distinct characters. This is true even in those instances where they correspond in semantics, pronunciation, or appearance.

Characters are represented by code points that reside only in a memory representation, as strings in memory, on disk, or in data transmission. The Unicode Standard deals only with character codes.

Glyphs represent the shapes that characters can have when they are rendered or displayed. In contrast to characters, glyphs appear on the screen or paper as particular representations of one or more characters. A repertoire of glyphs makes up a

font. Glyph shape and methods of identifying and selecting glyphs are the responsibility of individual font vendors and of appropriate standards and are not part of the Unicode Standard.

Various relationships may exist between character and glyph: a single glyph may correspond to a single character or to a number of characters, or multiple glyphs may result from a single character. The distinction between characters and glyphs is illustrated in *Figure 2-2*.

Figure 2-2. Characters Versus Glyphs

Glyphs	Unicode Characters
A A A A A A A A A	U+0041 LATIN CAPITAL LETTER A
a a a a a a a a	U+0061 LATIN SMALL LETTER A
п *n* *ū*	U+043F CYRILLIC SMALL LETTER PE
ه ه ه ه	U+0647 ARABIC LETTER HEH
fi fi	U+0066 LATIN SMALL LETTER F + U+0069 LATIN SMALL LETTER I

Even the letter "a" has a wide variety of glyphs that can represent it. A lowercase Cyrillic "п" also has a variety of glyphs; the second glyph for U+043F CYRILLIC SMALL LETTER PE shown in *Figure 2-2* is customary for italic in Russia, while the third is customary for italic in Serbia. Arabic letters are displayed with different glyphs, depending on their position in a word; the glyphs in *Figure 2-2* show independent, final, initial, and medial forms. Sequences such as "fi" may be displayed with two independent glyphs or with a ligature glyph.

What the user thinks of as a single character—which may or may not be represented by a single glyph—may be represented in the Unicode Standard as multiple code points. See *Table 2-2* for additional examples.

Table 2-2. User-Perceived Characters with Multiple Code Points

Character	Code Points	Linguistic Usage
ch	0063 0068	Slovak, traditional Spanish
tʰ	0074 02B0	Native American languages
x̣	0078 0323	
ƛ̓	019B 0313	
ą́	00E1 0328	Lithuanian
í	0069 0307 0301	
ト゚	30C8 309A	Ainu (in kana transcription)

For certain scripts, such as Arabic and the various Indic scripts, the number of glyphs needed to display a given script may be significantly larger than the number of characters encoding the basic units of that script. The number of glyphs may also depend on the orthographic style supported by the font. For example, an Arabic font intended to support the *Nastaliq* style of Arabic script may possess many thousands of glyphs. However, the character encoding employs the same few dozen letters regardless of the font style used to depict the character data in context.

A font and its associated rendering process define an arbitrary mapping from Unicode characters to glyphs. Some of the glyphs in a font may be independent forms for individual characters; others may be rendering forms that do not directly correspond to any single character.

Text rendering requires that characters in memory be mapped to glyphs. The final appearance of rendered text may depend on context (neighboring characters in the memory representation), variations in typographic design of the fonts used, and formatting information (point size, superscript, subscript, and so on). The results on screen or paper can differ considerably from the prototypical shape of a letter or character, as shown in <u>*Figure 2-3*</u>.

Figure 2-3. Unicode Character Code to Rendered Glyphs

Text Character Sequence

①	प	0000 1001 0010 1010
②	ॢ	0000 1001 0100 0010
③	र	0000 1001 0011 0000
④	ॢ	0000 1001 0100 1101
⑤	त	0000 1001 0010 0100
⑥	िॢ	0000 1001 0011 1111

Font
(Glyph Source)

Text
Rendering
Process

पूर्ति
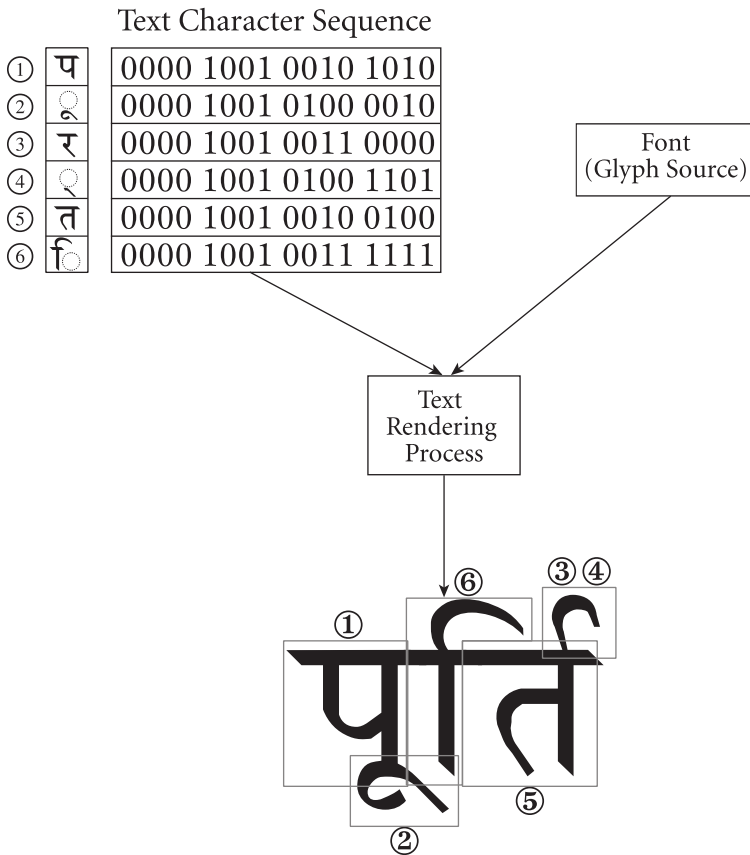
For the Latin script, this relationship between character code sequence and glyph is relatively simple and well known; for several other scripts, it is documented in this standard. However, in all cases, fine typography requires a more elaborate set of rules than given here. The Unicode Standard documents the default relationship between character sequences and glyphic appearance for the purpose of ensuring that the same text content can be stored with the same, and therefore interchangeable, sequence of character codes.

2.2.4 Semantics

Characters have well-defined semantics. These semantics are defined by explicitly assigned character properties, rather than implied through the character name or the position of a character in the code tables (see *Section 3.5, Properties*). The Unicode Character Database provides machine-readable character property tables for use in implementations of parsing, sorting, and other algorithms requiring semantic knowledge about the code points. These properties are supplemented by the description of

script and character behavior in this standard. See also Unicode Technical Report #23, "The Unicode Character Property Model."

The Unicode Standard identifies more than 100 different character properties, including numeric, casing, combination, and directionality properties (see *Chapter 4, Character Properties*). Additional properties may be defined as needed from time to time. Where characters are used in different ways in different languages, the relevant properties are normally defined outside the Unicode Standard. For example, Unicode Technical Standard #10, "Unicode Collation Algorithm," defines a set of default collation weights that can be used with a standard algorithm. Tailorings for each language are provided in the Unicode Common Locale Data Repository (CLDR); see https://cldr.unicode.org.

The Unicode Standard, by supplying a universal repertoire associated with well-defined character semantics, does not require the *code set independent* model of internationalization and text handling. That model abstracts away string handling as manipulation of byte streams of unknown semantics to protect implementations from the details of hundreds of different character encodings and selectively late-binds locale-specific character properties to characters. Of course, it is always possible for code set independent implementations to retain their model and to treat Unicode characters as just another character set in that context. It is not at all unusual for Unix implementations to simply add UTF-8 as another character set, parallel to all the other character sets they support. By contrast, the Unicode approach—because it is associated with a universal repertoire—assumes that characters and their properties are inherently and inextricably associated. If an internationalized application can be structured to work directly in terms of Unicode characters, all levels of the implementation can reliably and efficiently access character storage and be assured of the universal applicability of character property semantics.

2.2.5 Plain Text

Plain text is a pure sequence of character codes; plain Unicode-encoded text is therefore a sequence of Unicode character codes. In contrast, *styled text*, also known as *rich text*, is any text representation consisting of plain text plus added information such as a language identifier, font size, color, hypertext links, and so on. For example, the text of this specification, a multi-font text as formatted by a book editing system, is rich text.

The simplicity of plain text gives it a natural role as a major structural element of rich text. SGML, RTF, HTML, XML, and T_EX are examples of rich text fully represented as plain text streams, interspersing plain text data with sequences of characters that represent the additional data structures. They use special conventions embedded within the plain text file, such as "<p>", to distinguish the markup or *tags* from the

"real" content. Many popular word processing packages rely on a buffer of plain text to represent the content and implement links to a parallel store of formatting data.

The relative functional roles of both plain text and rich text are well established:

- Plain text is the underlying content stream to which formatting can be applied.
- Rich text carries complex formatting information as well as text context.
- Plain text is public, standardized, and universally readable.
- Rich text representation may be implementation-specific or proprietary.

Although some rich text formats have been standardized or made public, the majority of rich text designs are vehicles for particular implementations and are not necessarily readable by other implementations. Given that rich text equals plain text plus added information, the extra information in rich text can always be stripped away to reveal the "pure" text underneath. This operation is often employed, for example, in word processing systems that use both their own private rich text format and plain text file format as a universal, if limited, means of exchange. Thus, by default, plain text represents the basic, interchangeable content of text.

Plain text represents character content only, not its appearance. It can be displayed in a variety of ways and requires a rendering process to make it visible with a particular appearance. If the same plain text sequence is given to disparate rendering processes, there is no expectation that rendered text in each instance should have the same appearance. Instead, the disparate rendering processes are simply required to make the text legible according to the intended reading. This legibility criterion constrains the range of possible appearances. The relationship between appearance and content of plain text may be summarized as follows:

Plain text must contain enough information to permit the text to be rendered legibly, and nothing more.

The Unicode Standard encodes plain text. The distinction between plain text and other forms of data in the same data stream is the function of a higher-level protocol and is not specified by the Unicode Standard itself.

2.2.6 Logical Order

The order in which Unicode text is stored in the memory representation is called *logical order*. This order roughly corresponds to the order in which text is typed in via the keyboard; it also roughly corresponds to phonetic order. For decimal numbers, the logical order consistently corresponds to the most significant digit first, which is the order expected by number-parsing software.

When displayed, this logical order often corresponds to a simple linear progression of characters in one direction, such as from left to right, right to left, or top to bottom. In other circumstances, text is displayed or printed in an order that differs from a single linear progression. Some of the clearest examples are situations where a right-to-left script (such as Arabic or Hebrew) is mixed with a left-to-right script (such as Latin or Greek). For example, when the text in *Figure 2-4* is ordered for display the glyph that represents the first character of the English text appears at the left. The logical start character of the Hebrew text, however, is represented by the Hebrew glyph closest to the right margin. The succeeding Hebrew glyphs are laid out to the left.

Figure 2-4. Bidirectional Ordering

In logical order, numbers are encoded with most significant digit first, but are displayed in different writing directions. As shown in *Figure 2-5* these writing directions do not always correspond to the writing direction of the surrounding text. The first example shows N'Ko, a right-to-left script with digits that also render right to left. Examples 2 and 3 show Hebrew and Arabic, in which the numbers are rendered left to right, resulting in bidirectional layout. In left-to-right scripts, such as Latin and Hiragana and Katakana (for Japanese), numbers follow the predominant left-to-right direction of the script, as shown in Examples 4 and 5. When Japanese is laid out vertically, numbers are either laid out vertically or may be rotated clockwise ninety degrees to follow the layout direction of the lines, as shown in Example 6.

Figure 2-5. Writing Direction and Numbers

The Unicode Standard precisely defines the conversion of Unicode text from logical order to the order of readable (displayed) text so as to ensure consistent legibility. Properties of directionality inherent in characters generally determine the correct display order of text. The Unicode Bidirectional Algorithm specifies how these properties are used to resolve directional interactions when characters of right-to-left and left-to-right directionality are mixed. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm.") However, when characters of different directionality are

mixed, inherent directionality alone is occasionally insufficient to render plain text legibly. The Unicode Standard therefore includes characters to explicitly specify changes in direction when necessary. The Bidirectional Algorithm uses these directional layout control characters together with the inherent directional properties of characters to exert exact control over the display ordering for legible interchange. By requiring the use of this algorithm, the Unicode Standard ensures that plain text used for simple items like file names or labels can always be correctly ordered for display.

Besides mixing runs of differing overall text direction, there are many other cases where the logical order does not correspond to a linear progression of characters. Combining characters (such as accents) are stored following the base character to which they apply, but are positioned relative to that base character and thus do not follow a simple linear progression in the final rendered text. For example, the Latin letter "x̱" is stored as "x" followed by combining "◌̱"; the accent appears below, not to the right of the base. This position with respect to the base holds even where the overall text progression is from top to bottom—for example, with "x̱" appearing upright within a vertical Japanese line. Characters may also combine into ligatures or conjuncts or otherwise change positions of their components radically, as shown in *Figure 2-3* and *Figure 2-19*.

There is one particular exception to the usual practice of logical order paralleling phonetic order. With the Thai, Lao, Tai Viet, and New Tai Lue scripts, users traditionally type in visual order rather than phonetic order, resulting in some vowel letters being stored ahead of consonants, even though they are pronounced after them.

2.2.7 Unification

The Unicode Standard avoids duplicate encoding of characters by unifying them within scripts across language. Common letters are given one code each, regardless of language, as are common Chinese/Japanese/Korean (CJK) ideographs. (See *Section 18.1, Han*.)

Punctuation marks, symbols, and diacritics are handled in a similar manner as letters. If they can be clearly identified with a particular script, they are encoded once for that script and are unified across any languages that may use that script. See, for example, U+1362 ETHIOPIC FULL STOP, U+060F ARABIC SIGN MISRA, and U+0592 HEBREW ACCENT SEGOL. However, some punctuation or diacritical marks may be shared in common across a number of scripts—the obvious example being Western-style punctuation characters, which are often recently added to the writing systems of scripts other than Latin. In such cases, characters are encoded only once and are intended for use with multiple scripts. Common symbols are also encoded only once and are not associated with any script in particular.

It is quite normal for many characters to have different usages, such as *comma* " , " for either thousands-separator (English) or decimal-separator (French). The Unicode

Standard avoids duplication of characters due to specific usage in different languages; rather, it duplicates characters *only* to support compatibility with base standards. Avoidance of duplicate encoding of characters is important to avoid visual ambiguity.

There are a few notable instances in the standard where visual ambiguity between different characters is tolerated, however. For example, in most fonts there is little or no distinction visible between Latin "o", Cyrillic "o", and Greek "o" (*omicron)*. These are not unified because they are characters from three different scripts, and many legacy character encodings distinguish between them. As another example, there are three characters whose glyph is the same uppercase barred D shape, but they correspond to three distinct lowercase forms. Unifying these uppercase characters would have resulted in unnecessary complications for case mapping.

The Unicode Standard does not attempt to encode features such as language, font, size, positioning, glyphs, and so forth. For example, it does not preserve language as a part of character encoding: just as French *i grec*, German *ypsilon*, and English *wye* are all represented by the same character code, U+0059 "Y", so too are Chinese *zi*, Japanese *ji,* and Korean *ja* all represented as the same character code, U+5B57 字.

In determining whether to unify variant CJK ideograph forms across standards, the Unicode Standard follows the principles described in *Section 18.1, Han*. Where these principles determine that two forms constitute a trivial difference, the Unicode Standard assigns a single code. Just as for the Latin and other scripts, typeface distinctions or local preferences in glyph shapes alone are not sufficient grounds for disunification of a character. *Figure 2-6* illustrates the well-known example of the CJK ideograph for "bone," which shows significant shape differences from typeface to typeface, with some forms preferred in China and some in Japan. All of these forms are considered to be the same *character*, encoded at U+9AA8 in the Unicode Standard.

Figure 2-6. Typeface Variation for the Bone Character

骨 骨

Many characters in the Unicode Standard could have been unified with existing visually similar Unicode characters or could have been omitted in favor of some other Unicode mechanism for maintaining the kinds of text distinctions for which they were intended. However, considerations of interoperability with other standards and systems often require that such compatibility characters be included in the Unicode Standard. See *Section 2.3, Compatibility Characters*. In particular, whenever font style, size, positioning or precise glyph shape carry a specific meaning and are

used in distinction to the ordinary character—for example, in phonetic or mathematical notation—the characters are not unified.

2.2.8 Dynamic Composition

The Unicode Standard allows for the dynamic composition of accented forms and Hangul syllables. Combining characters used to create composite forms are productive. Because the process of character composition is open-ended, new forms with modifying marks may be created from a combination of base characters followed by combining characters. For example, the diaeresis """ may be combined with all vowels and a number of consonants in languages using the Latin script and several other scripts, as shown in *Figure 2-7*.

Figure 2-7. Dynamic Composition

$$A + \overset{..}{\circ} \rightarrow \ddot{A}$$

0041   0308

Equivalent Sequences. Some text elements can be encoded either as static precomposed forms or by dynamic composition. Common precomposed forms such as U+00DC "Ü" LATIN CAPITAL LETTER U WITH DIAERESIS are included for compatibility with current standards. For static precomposed forms, the standard provides a mapping to an equivalent dynamically composed sequence of characters. (See also *Section 3.7, Decomposition*.) Thus different sequences of Unicode characters are considered equivalent. A precomposed character may be represented as an equivalent composed character sequence (see *Section 2.12, Equivalent Sequences*).

2.2.9 Stability

Certain aspects of the Unicode Standard must be absolutely stable between versions, so that implementers and users can be guaranteed that text data, once encoded, retains the same meaning. Most importantly, this means that once Unicode characters are assigned, their code point assignments cannot be changed, nor can characters be removed.

Characters are retained in the standard, so that previously conforming data stay conformant in future versions of the standard. Sometimes characters are deprecated —that is, their use in new documents is strongly discouraged. While implementations should continue to recognize such characters when they are encountered, spellcheckers or editors could warn users of their presence and suggest replacements. For more about deprecated characters, see D13 in *Section 3.4, Characters and Encoding*.

Unicode character names are also never changed, so that they can be used as identifiers that are valid across versions. See *Section 4.8, Name*.

Similar stability guarantees exist for certain important properties. For example, the decompositions are kept stable, so that it is possible to normalize a Unicode text once and have it remain normalized in all future versions.

The most current versions of the character encoding stability policies for the Unicode Standard are maintained online at:

https://www.unicode.org/policies/stability_policy.html

2.2.10 Convertibility

Character identity is preserved for interchange with a number of different base standards, including national, international, and vendor standards. Where variant forms (or even the same form) are given separate codes within one base standard, they are also kept separate within the Unicode Standard. This choice guarantees the existence of a mapping between the Unicode Standard and base standards.

Accurate convertibility is guaranteed between the Unicode Standard and other standards in wide usage as of May 1993. Characters have also been added to allow convertibility to several important East Asian character sets created after that date—for example, GB 18030. In general, a single code point in another standard will correspond to a single code point in the Unicode Standard. Sometimes, however, a single code point in another standard corresponds to a sequence of code points in the Unicode Standard, or vice versa. Conversion between Unicode text and text in other character codes must, in general, be done by explicit table-mapping processes. (See also *Section 5.1, Data Structures for Character Conversion*.)

2.3 Compatibility Characters

Conceptually, compatibility characters are characters that would not have been encoded in the Unicode Standard except for compatibility and round-trip convertibility with other standards. Such standards include international, national, and vendor character encoding standards. For the most part, these are widely used standards that pre-dated Unicode, but because continued interoperability with new standards and data sources is one of the primary design goals of the Unicode Standard, additional compatibility characters are added as the situation warrants.

Compatibility characters can be contrasted with *ordinary* (or non-compatibility) characters in the standard—ones that are generally consistent with the Unicode text model and which would have been accepted for encoding to represent various scripts and sets of symbols, regardless of whether those characters also existed in other character encoding standards.

For example, in the Unicode model of Arabic text the logical representation of text uses basic Arabic letters. Rather than being directly represented in the encoded characters, the cursive presentation of Arabic text for display is determined in context by a rendering system. (See *Section 9.2, Arabic*.) However, some earlier character encodings for Arabic were intended for use with rendering systems that required separate characters for initial, medial, final, and isolated presentation forms of Arabic letters. To allow one-to-one mapping to these character sets, the Unicode Standard includes Arabic presentation forms as compatibility characters.

The purpose for the inclusion of compatibility characters like these is not to implement or emulate alternative text models, nor to encourage the use of plain text distinctions in characters which would otherwise be better represented by higher-level protocols or other mechanisms. Rather, the main function of compatibility characters is to simplify interoperability of Unicode-based systems with other data sources, and to ensure convertibility of data.

Interoperability does not require that all external characters can be mapped to single Unicode characters; encoding a compatibility character is not necessary when a character in another standard can be represented as a sequence of existing Unicode characters. For example the Shift-JIS encoding 0x839E for JIS X 0213 *katakana letter ainu to* can simply be mapped to the Unicode character sequence <U+30C8, U+309A>. However, in cases where no appropriate mapping is available, the requirement for interoperability and convertibility may be met by encoding a compatibility character for one-to-one mapping to another standard.

Usage. The fact that a particular character is considered a compatibility character does not mean that that character is deprecated in the standard. The use of most compatibility characters in general text interchange is unproblematic. Some,

however, such as the Arabic positional forms or other compatibility characters which assume information about particular layout conventions, such as presentation forms for vertical text, can lead to problems when used in general interchange. Caution is advised for their use. See also the discussion of compatibility characters in the W3C specification, "Unicode in XML and Other Markup Languages."

Allocation. The Compatibility and Specials Area contains a large number of compatibility characters, but the Unicode Standard also contains many compatibility characters that do not appear in that area. These include examples such as U+2163 "IV" ROMAN NUMERAL FOUR, U+2007 FIGURE SPACE, U+00B2 "²" SUPERSCRIPT TWO, U+2502 BOX DRAWINGS LIGHT VERTICAL, and U+32D0 CIRCLED KATAKANA A.

There is no formal listing of all compatibility characters in the Unicode Standard. This follows from the nature of the definition of compatibility characters. It is a judgement call as to whether any particular character would have been accepted for encoding if it had not been required for interoperability with a particular standard. Different participants in character encoding often disagree about the appropriateness of encoding particular characters, and sometimes there are multiple justifications for encoding a given character.

2.3.1 Compatibility Variants

Compatibility variants are a subset of compatibility characters, and have the further characteristic that they represent variants of existing, ordinary, Unicode characters.

For example, compatibility variants might represent various presentation or styled forms of basic letters: superscript or subscript forms, variant glyph shapes, or vertical presentation forms. They also include halfwidth or fullwidth characters from East Asian character encoding standards, Arabic contextual form glyphs from preexisting Arabic code pages, Arabic ligatures and ligatures from other scripts, and so on. Compatibility variants also include CJK compatibility ideographs, many of which are minor glyph variants of an encoded unified CJK ideograph.

In contrast to compatibility variants there are the numerous compatibility characters, such as U+2502 BOX DRAWINGS LIGHT VERTICAL, U+263A WHITE SMILING FACE, or U+2701 UPPER BLADE SCISSORS, which are not variants of ordinary Unicode characters. However, it is not always possible to determine unequivocally whether a compatibility character is a variant or not.

2.3.2 Compatibility Decomposable Characters

The term *compatibility* is further applied to Unicode characters in a different, strictly defined sense. The concept of a *compatibility decomposable character* is formally defined as any Unicode character whose compatibility decomposition is not identical

to its canonical decomposition. (See Definition D66 in *Section 3.7, Decomposition*, and the discussion in *Section 2.2, Unicode Design Principles*.)

The list of compatibility decomposable characters is precisely defined by property values in the Unicode Character Database, and by the rules of Unicode Normalization. (See *Section 3.11, Normalization Forms*.) Because of their use in Unicode Normalization, compatibility decompositions are stable and cannot be changed once a character has been encoded; the list of compatibility decomposable characters for any version of the Unicode Standard is thus also stable.

Compatibility decomposable characters have also been referred to in earlier versions of the Unicode Standard as *compatibility composite characters* or *compatibility composites* for short, but the full term, *compatibility decomposable character* is preferred.

Compatibility Character Vs. Compatibility Decomposable Character. In informal discussions of the Unicode Standard, compatibility decomposable characters have also often been referred to simply as "compatibility characters." This is understandable, in part because the two sets of characters largely overlap, but the concepts are actually distinct. There are compatibility characters which are not compatibility decomposable characters, and there are compatibility decomposable characters which are not compatibility characters.

For example, the deprecated alternate format characters such as U+206C INHIBIT ARABIC FORM SHAPING are considered compatibility characters, but they have no decomposition mapping, and thus by definition cannot be compatibility decomposable characters. Likewise for such other compatibility characters as U+2502 BOX DRAWINGS LIGHT VERTICAL or U+263A WHITE SMILING FACE.

There are also instances of compatibility variants which clearly *are* variants of other Unicode characters, but which have no decomposition mapping. For example, U+2EAF CJK RADICAL SILK is a compatibility variant of U+2F77 KANGXI RADICAL SILK, as well as being a compatibility variant of U+7CF9 CJK UNIFIED IDEOGRAPH-7CF9, but has no compatibility decomposition. The numerous compatibility variants like this in the CJK Radicals Supplement block were encoded for compatibility with encodings that distinguished and separately encoded various forms of CJK radicals as symbols.

A different case is illustrated by the CJK compatibility ideographs, such as U+FA0C CJK COMPATIBILITY IDEOGRAPH-FA0C. Those compatibility characters have a decomposition mapping, but for historical reasons it is always a canonical decomposition, so they are canonical decomposable characters, but *not* compatibility decomposable characters.

By way of contrast, some compatibility decomposable characters, such as modifier letters used in phonetic orthographies, for example, U+02B0 MODIFIER LETTER

SMALL H, are *not* considered to be compatibility characters. They would have been accepted for encoding in the standard on their own merits, regardless of their need for mapping to IPA. A large number of compatibility decomposable characters like this are actually distinct symbols used in specialized notations, whether phonetic or mathematical. In such cases, their compatibility mappings express their historical derivation from styled forms of standard letters.

Other compatibility decomposable characters are widely used characters serving essential functions. U+00A0 NO-BREAK SPACE is one example. In these and similar cases, such as fixed-width space characters, the compatibility decompositions define possible fallback representations.

The Unicode Character Database supplies identification and mapping information only for compatibility decomposable characters, while compatibility variants are not formally identified or documented. Because the two sets substantially overlap, many specifications are written in terms of compatibility decomposable characters first; if necessary, such specifications may be extended to handle other, non-decomposable compatibility variants as required. (See also the discussion in *Section 5.19, Mapping Compatibility Variants.*)

2.4 Code Points and Characters

On a computer, abstract characters are encoded internally as numbers. To create a complete character encoding, it is necessary to define the list of all characters to be encoded and to establish systematic rules for how the numbers represent the characters.

The range of integers used to code the abstract characters is called the *codespace*. A particular integer in this set is called a *code point*. When an abstract character is mapped or *assigned* to a particular code point in the codespace, it is then referred to as an *encoded character*.

In the Unicode Standard, the codespace consists of the integers from 0 to $10FFFF_{16}$, comprising 1,114,112 code points available for assigning the repertoire of abstract characters.

There are constraints on how the codespace is organized, and particular areas of the codespace have been set aside for encoding of certain kinds of abstract characters or for other uses in the standard. For more on the *allocation* of the Unicode codespace, see *Section 2.8, Unicode Allocation*.

Figure 2-8 illustrates the relationship between abstract characters and code points, which together constitute encoded characters. Note that some abstract characters may be associated with multiple, separately encoded characters (that is, be encoded "twice"). In other instances, an abstract character may be represented by a sequence of two (or more) other encoded characters. The solid arrows connect encoded characters with the abstract characters that they represent and encode.

Figure 2-8. Abstract and Encoded Characters
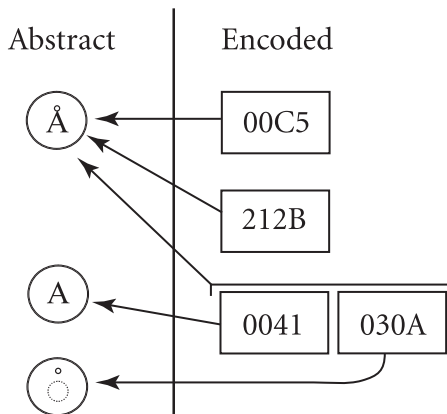
When referring to code points in the Unicode Standard, the usual practice is to refer to them by their numeric value expressed in hexadecimal, with a "U+" prefix. (See *Appendix A, Notational Conventions*.) Encoded characters can also be referred to by their code points only. To prevent ambiguity, the official Unicode name of the character is often added; this clearly identifies the abstract character that is encoded. For example:

U+0061 LATIN SMALL LETTER A

U+10330 GOTHIC LETTER AHSA

U+201DF CJK UNIFIED IDEOGRAPH-201DF

Such citations refer only to the encoded character per se, associating the code point (as an integral value) with the abstract character that is encoded.

2.4.1 Types of Code Points

There are many ways to categorize code points. *Table 2-3* illustrates some of the categorizations and basic terminology used in the Unicode Standard. The seven basic types of code points are formally defined in *Section 3.4, Characters and Encoding*. (See Definition D10a, Code Point Type.)

Table 2-3. Types of Code Points

Basic Type	Brief Description	General Category	Character Status	Code Point Status
Graphic	Letter, mark, number, punctuation, symbol, and spaces	L, M, N, P, S, Zs	*Assigned to abstract character*	*Designated (assigned) code point*
Format	Invisible but affects neighboring characters; includes line/paragraph separators	Cf, Zl, Zp		
Control	Usage defined by protocols or standards outside the Unicode Standard	Cc		
Private-use	Usage defined by private agreement outside the Unicode Standard	Co		
Surrogate	Permanently reserved for UTF-16; restricted interchange	Cs	*Cannot be assigned to abstract character*	
Noncharacter	Permanently reserved for internal usage; restricted interchange	Cn	*Not assigned to abstract character*	

| Reserved | Reserved for future assignment; restricted interchange | | | *Undesignated (unassigned) code point* |

Not all assigned code points represent abstract characters; only Graphic, Format, Control and Private-use do. Surrogates and Noncharacters are assigned code points but are not assigned to abstract characters. Reserved code points are assignable: any may be assigned in a future version of the standard. The General Category provides a finer breakdown of Graphic characters and also distinguishes between the other basic types (except between Noncharacter and Reserved). Other properties defined in the Unicode Character Database provide for different categorizations of Unicode code points.

Control Codes. Sixty-five code points (U+0000..U+001F and U+007F..U+009F) are defined specifically as control codes, for compatibility with the C0 and C1 control codes of the ISO/IEC 2022 framework. A few of these control codes are given specific interpretations by the Unicode Standard. (See *Section 23.1, Control Codes*.)

Noncharacters. Sixty-six code points are not used to encode characters. Noncharacters consist of U+FDD0..U+FDEF and any code point ending in the value $FFFE_{16}$ or $FFFF_{16}$—that is, U+FFFE, U+FFFF, U+1FFFE, U+1FFFF, ... U+10FFFE, U+10FFFF. (See *Section 23.7, Noncharacters*.)

Private Use. Three ranges of code points have been set aside for private use. Characters in these areas will never be defined by the Unicode Standard. These code points can be freely used for characters of any purpose, but successful interchange requires an agreement between sender and receiver on their interpretation. (See *Section 23.5, Private-Use Characters*.)

Surrogates. Some 2,048 code points have been allocated as surrogate code points, which are used in the UTF-16 encoding form. (See *Section 23.6, Surrogates Area*.)

Restricted Interchange. Code points that are not assigned to abstract characters are subject to restrictions in interchange.

- Surrogate code points cannot be conformantly interchanged using Unicode encoding forms. They do not correspond to Unicode scalar values and thus do not have well-formed representations in any Unicode encoding form. (See *Section 3.8, Surrogates*.)

- Noncharacter code points are reserved for internal use, such as for sentinel values. They have well-formed representations in Unicode encoding forms and survive conversions between encoding forms. This allows sentinel values to be preserved internally across Unicode encoding forms, even though they are not designed to be used in open interchange.

- All implementations need to preserve reserved code points because they may originate in implementations that use a *future* version of the Unicode Standard. For example, suppose that one person is using a Unicode 12.0 system and a second person is using a Unicode 11.0 system. The first person sends the second person a document containing some code points newly assigned in Unicode 12.0; these code points were unassigned in Unicode 11.0. The second person may edit the document, not changing the reserved codes, and send it on. In that case the second person is interchanging what are, as far as the second person knows, reserved code points.

Code Point Semantics. The semantics of most code points are established by this standard; the exceptions are Controls, Private-use, and Noncharacters. Control codes generally have semantics determined by other standards or protocols (such as ISO/IEC 6429), but there are a small number of control codes for which the Unicode Standard specifies particular semantics. See *Table 23-1* in *Section 23.1, Control Codes*, for the exact list of those control codes. The semantics of private-use characters are outside the scope of the Unicode Standard; their use is determined by private agreement, as, for example, between vendors. Noncharacters have semantics in internal use only.

2.5 Encoding Forms

Computers handle numbers not simply as abstract mathematical objects, but as combinations of fixed-size units like bytes and 32-bit words. A character encoding model must take this fact into account when determining how to associate numbers with the characters.

Actual implementations in computer systems represent integers in specific *code units* of particular size—usually 8-bit (= byte), 16-bit, or 32-bit. In the Unicode character encoding model, precisely defined *encoding forms* specify how each integer (code point) for a Unicode character is to be expressed as a sequence of one or more code units. The Unicode Standard provides three distinct encoding forms for Unicode characters, using 8-bit, 16-bit, and 32-bit units. These are named UTF-8, UTF-16, and UTF-32, respectively. The "UTF" is a carryover from earlier terminology meaning Unicode (or UCS) Transformation Format. Each of these three encoding forms is an equally legitimate mechanism for representing Unicode characters; each has advantages in different environments.

All three encoding forms can be used to represent the full range of encoded characters in the Unicode Standard; they are thus fully interoperable for implementations that may choose different encoding forms for various reasons. Each of the three Unicode encoding forms can be efficiently transformed into either of the other two without any loss of data.

Non-overlap. Each of the Unicode encoding forms is designed with the principle of non-overlap in mind. *Figure 2-9* presents an example of an encoding where overlap is permitted. In this encoding (Windows code page 932), characters are formed from either one or two code bytes. Whether a sequence is one or two bytes in length depends on the first byte, so that the values for lead bytes (of a two-byte sequence) and single bytes are disjoint. However, single-byte values and trail-byte values can overlap. That means that when someone searches for the character "D", for example, he or she might find it either (mistakenly) as the trail byte of a two-byte sequence or as a single, independent byte. To find out which alternative is correct, a program must look backward through text.

Figure 2-9. Overlap in Legacy Mixed-Width Encodings
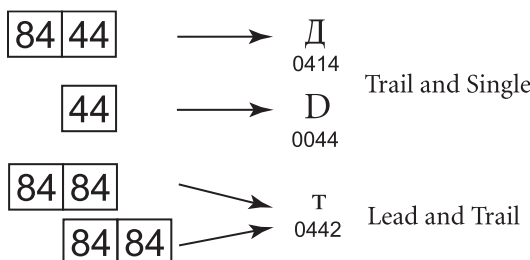
The situation is made more complex by the fact that lead and trail bytes can also overlap, as shown in the second part of *Figure 2-9*. This means that the backward scan has to repeat until it hits the start of the text or hits a sequence that could not exist as a pair as shown in *Figure 2-10*. This is not only inefficient, but also extremely error-prone: corruption of one byte can cause entire lines of text to be corrupted.

Figure 2-10. Boundaries and Interpretation
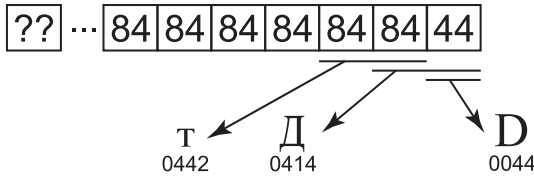
The Unicode encoding forms avoid this problem, because *none* of the ranges of values for the lead, trail, or single code units in any of those encoding forms overlap.

Non-overlap makes all of the Unicode encoding forms well behaved for searching and comparison. When searching for a particular character, there will never be a mismatch against some code unit sequence that represents just part of another character. The fact that all Unicode encoding forms observe this principle of non-overlap distinguishes them from many legacy East Asian multibyte character encodings, for which overlap of code unit sequences may be a significant problem for implementations.

Another aspect of non-overlap in the Unicode encoding forms is that all Unicode characters have determinate boundaries when expressed in any of the encoding forms. That is, the edges of code unit sequences representing a character are easily determined by local examination of code units; there is never any need to scan back indefinitely in Unicode text to correctly determine a character boundary. This property of the encoding forms has sometimes been referred to as *self-synchronization*. This property has another very important implication: corruption of a single code unit corrupts *only* a single character; none of the surrounding characters are affected.

For example, when randomly accessing a string, a program can find the boundary of a character with limited backup. In UTF-16, if a pointer points to a leading surrogate, a single backup is required. In UTF-8, if a pointer points to a byte starting with 10xxxxxx (in binary), one to three backups are required to find the beginning of the character.

Conformance. The Unicode Consortium fully endorses the use of any of the three Unicode encoding forms as a conformant way of implementing the Unicode Standard. It is important not to fall into the trap of trying to distinguish "UTF-8 *versus* Unicode," for example. UTF-8, UTF-16, and UTF-32 are *all* equally valid and conformant ways of implementing the encoded characters of the Unicode Standard.

Examples. *Figure 2-11* shows the three Unicode encoding forms, including how they are related to Unicode code points.

Figure 2-11. Unicode Encoding Forms

In *Figure 2-11*, the UTF-32 line shows that each example character can be expressed with one 32-bit code unit. Those code units have the same values as the code point for the character. For UTF-16, most characters can be expressed with one 16-bit code unit, whose value is the same as the code point for the character, but characters with high code point values require a pair of 16-bit surrogate code units instead. In UTF-8, a character may be expressed with one, two, three, or four bytes, and the relationship between those byte values and the code point value is more complex.

UTF-8, UTF-16, and UTF-32 are further described in the subsections that follow. See each subsection for a general overview of how each encoding form is structured and the general benefits or drawbacks of each encoding form for particular purposes. For the detailed formal definition of the encoding forms and conformance requirements, see *Section 3.9, Unicode Encoding Forms*.

2.5.1 UTF-32

UTF-32 is the simplest Unicode encoding form. Each Unicode code point is represented directly by a single 32-bit code unit. Because of this, UTF-32 has a one-to-one relationship between encoded character and code unit; it is a fixed-width character encoding form.

Implementations that generally use UTF-8 or UTF-16 sometimes temporarily convert strings to UTF-32 for easier processing.

In some use cases, multi-code point sequences are useful or necessary as units of processing; for example, grapheme clusters, or sequences of characters with non-zero combining classes. This may limit the usefulness of a per-code point fixed-width encoding.

String encoding forms like UTF-32 or UTF-8 are irrelevant for APIs that pass single character values: These typically take or return simple code point integers.

As for all of the Unicode encoding forms, UTF-32 is restricted to representation of code points in the ranges $0..D7FF_{16}$ and $E000_{16}..10FFFF_{16}$—that is, Unicode scalar values. This guarantees interoperability with the UTF-16 and UTF-8 encoding forms.

Fixed Width. The value of each UTF-32 code unit corresponds exactly to the Unicode code point value. This situation differs significantly from that for UTF-16 and especially UTF-8, where the code unit values often change unrecognizably from the code point value. For example, U+10000 is represented as <00010000> in UTF-32 and as <F0 90 80 80> in UTF-8. For UTF-32, it is trivial to determine a Unicode character from its UTF-32 code unit representation. In contrast, UTF-16 and UTF-8 representations often require doing a code unit conversion before the character can be identified in the Unicode code charts.

Preferred Usage. UTF-32 may be a preferred encoding form where memory or disk storage space for characters is not a particular concern, but where fixed-width, single code unit access to characters is desired. For example, Python 3 strings are sequences of Unicode code points.

2.5.2 UTF-16

In the UTF-16 encoding form, non-surrogate code points in the range U+0000..U+FFFF are represented as a single 16-bit code unit; code points in the supplementary planes, in the range U+10000..U+10FFFF, are represented as pairs of 16-bit code units. These pairs of special code units are known as *surrogate pairs*. The values of the code units used for surrogate pairs are completely disjunct from the code units used for the single code unit representations, thus maintaining non-overlap for all code point representations in UTF-16. For the formal definition of surrogates, see *Section 3.8, Surrogates*.

Optimized for BMP. UTF-16 optimizes the representation of characters in the Basic Multilingual Plane (BMP)—that is, characters in the range U+0000..U+FFFF, excluding surrogate code points. For that range, which contains the vast majority of common-use characters for all modern scripts of the world, each character requires only one 16-bit code unit, thus requiring just half the memory or storage of the UTF-32 encoding form. For the BMP, UTF-16 can effectively be treated as if it were a fixed-width encoding form.

Supplementary Characters and Surrogates. For supplementary characters, UTF-16 requires two 16-bit code units. The distinction between characters represented with one versus two 16-bit code units means that UTF-16 is a variable-width encoding form. That fact can create implementation difficulties if it is not carefully taken into account; UTF-16 is somewhat more complicated to handle than UTF-32.

Preferred Usage. UTF-16 may be a preferred encoding form in many environments that need to balance efficient access to characters with economical use of storage. It

is reasonably compact, and all the common, heavily used characters fit into a single 16-bit code unit.

Origin. UTF-16 is the historical descendant of the earliest form of Unicode, which was originally designed to use a fixed-width, 16-bit encoding form exclusively. The surrogates were added to provide an encoding form for the supplementary characters at code points past U+FFFF. The design of the surrogates made them a simple and efficient extension mechanism that works well with older Unicode implementations and that avoids many of the problems of other variable-width character encodings. See *Section 5.4, Handling Surrogate Pairs in UTF-16*, for more information about surrogates and their processing.

Binary Sorting. For the purpose of sorting text, if the text contains supplementary code points, binary order for data represented in the UTF-16 encoding form is not the same as code point order. This means that a slightly different comparison implementation is needed for code point order. For more information, see *Section 5.17, Binary Order*.

2.5.3 UTF-8

To meet the requirements of byte-oriented, ASCII-based systems, a third encoding form is specified by the Unicode Standard: UTF-8. This variable-width encoding form preserves ASCII transparency by making use of 8-bit code units.

Byte-Oriented. Much existing software and practice in information technology have long depended on character data being represented as a sequence of bytes. Furthermore, many of the protocols depend not only on ASCII values being invariant, but must make use of or avoid special byte values that may have associated control functions. The easiest way to adapt Unicode implementations to such a situation is to make use of an encoding form that is already defined in terms of 8-bit code units and that represents all Unicode characters while not disturbing or reusing any ASCII or C0 control code value. That is the function of UTF-8.

Variable Width. UTF-8 is a variable-width encoding form, using 8-bit code units, in which the high bits of each code unit indicate the part of the code unit sequence to which each byte belongs. A range of 8-bit code unit values is reserved for the first, or *leading,* element of a UTF-8 code unit sequences, and a completely disjunct range of 8-bit code unit values is reserved for the subsequent, or *trailing,* elements of such sequences; this convention preserves non-overlap for UTF-8. *Table 3-6* shows how the bits in a Unicode code point are distributed among the bytes in the UTF-8 encoding form. See *Section 3.9, Unicode Encoding Forms*, for the full, formal definition of UTF-8.

ASCII Transparency. The UTF-8 encoding form maintains transparency for all of the ASCII code points (0x00..0x7F). That means Unicode code points

U+0000..U+007F are converted to single bytes 0x00..0x7F in UTF-8 and are thus indistinguishable from ASCII itself. Furthermore, the values 0x00..0x7F do not appear in any byte for the representation of any other Unicode code point, so that there can be no ambiguity. Beyond the ASCII range of Unicode, many of the non-ideographic scripts are represented by two bytes per code point in UTF-8; all non-surrogate code points between U+0800 and U+FFFF are represented by three bytes; and supplementary code points above U+FFFF require four bytes.

Data Size. UTF-8 is reasonably compact in terms of the number of bytes used. Compared with UTF-16, it is much smaller for ASCII syntax and Western languages, but significantly larger for Asian writing systems such as for Hindi, Thai, Chinese, Japanese, and Korean.

Preferred Usage. UTF-8 is typically the preferred encoding form for HTML and similar protocols, particularly for the Internet. The ASCII transparency helps migration. UTF-8 also has the advantage that it is already inherently byte-serialized, as for most existing 8-bit character sets; strings of UTF-8 work easily with the C standard library, and many existing APIs that work for typical East Asian multibyte character sets adapt to UTF-8 as well with little or no change required.

Self-synchronizing. In environments where 8-bit character processing is required for one reason or another, UTF-8 has the following attractive features as compared to other multibyte encodings:

- The first byte of a UTF-8 code unit sequence indicates the number of bytes to follow in a multibyte sequence. This allows for very efficient forward parsing.
- It is efficient to find the start of a character when beginning from an arbitrary location in a byte stream of UTF-8. Programs need to search at most four bytes backward, and usually much less. It is a simple task to recognize an initial byte, because initial bytes are constrained to a fixed range of values.
- As with the other encoding forms, there is no overlap of byte values.

Binary Sorting. A binary sort of UTF-8 strings gives the same ordering as a binary sort of Unicode code points. This is obviously the same order as for a binary sort of UTF-32 strings.

2.6 Encoding Schemes

The discussion of Unicode encoding forms in the previous section was concerned with the machine representation of Unicode code units. Each code unit is represented in a computer simply as a numeric data type; just as for other numeric types, the exact way the bits are laid out internally is irrelevant to most processing. However, interchange of textual data, particularly between computers of different architectural types, requires consideration of the exact ordering of the bits and bytes involved in numeric representation. Integral data, including character data, is *serialized* for open interchange into well-defined sequences of bytes. This process of *byte serialization* allows all applications to correctly interpret exchanged data and to accurately reconstruct numeric values (and thereby character values) from it. In the Unicode Standard, the specifications of the distinct types of byte serializations to be used with Unicode data are known as Unicode *encoding schemes*.

Byte Order. Modern computer architectures differ in *ordering* in terms of whether the most significant byte or the least significant byte of a large numeric data type comes first in internal representation. These sequences are known as "big-endian" and "little-endian" orders, respectively. For the Unicode 16- and 32-bit encoding forms (UTF-16 and UTF-32), the specification of a byte serialization must take into account the big-endian or little-endian architecture of the system on which the data is represented, so that when the data is byte serialized for interchange it will be well defined.

A *character encoding scheme* consists of a specified character encoding form plus a specification of how the code units are serialized into bytes. The Unicode Standard also specifies the use of an initial *byte order mark* (BOM) to explicitly differentiate big-endian or little-endian data in some of the Unicode encoding schemes. (See the "Byte Order Mark" subsection in *Section 23.8, Specials*.)

When a higher-level protocol supplies mechanisms for handling the endianness of integral data types, it is not necessary to use Unicode encoding schemes or the byte order mark. In those cases Unicode text is simply a sequence of integral data types.

For UTF-8, the encoding scheme consists merely of the UTF-8 code units (= bytes) in sequence. Hence, there is no issue of big- versus little-endian byte order for data represented in UTF-8. However, for 16-bit and 32-bit encoding forms, byte serialization must break up the code units into two or four bytes, respectively, and the order of those bytes must be clearly defined. Because of this, and because of the rules for the use of the byte order mark, the three encoding forms of the Unicode Standard result in a total of seven Unicode encoding schemes, as shown in *Table 2-4*.

Table 2-4. The Seven Unicode Encoding Schemes

Encoding Scheme	Endian Order	BOM Allowed?
UTF-8	N/A	yes
UTF-16	Big-endian or little-endian	yes
UTF-16BE	Big-endian	no
UTF-16LE	Little-endian	no
UTF-32	Big-endian or little-endian	yes
UTF-32BE	Big-endian	no
UTF-32LE	Little-endian	no

The endian order entry for UTF-8 in *Table 2-4* is marked N/A because UTF-8 code units are 8 bits in size, and the usual machine issues of endian order for larger code units do not apply. The serialized order of the bytes must not depart from the order defined by the UTF-8 encoding form. Use of a BOM is not required for UTF-8, but may be encountered in contexts where UTF-8 data is converted from other encoding forms that use a BOM or where the BOM is used as a UTF-8 signature. See the "Byte Order Mark" subsection in *Section 23.8, Specials*, for more information. The Unicode Standard does not recommend for or against use of a BOM in UTF-8 data.

Encoding Scheme Versus Encoding Form. Note that some of the Unicode encoding schemes have the same labels as the three Unicode encoding forms. This could cause confusion, so it is important to keep the context clear when using these terms: character encoding *forms* refer to integral data units in memory or in APIs, and byte order is irrelevant; character encoding *schemes* refer to byte-serialized data, as for streaming I/O or in file storage, and byte order *must* be specified or determinable.

The Internet Assigned Numbers Authority (IANA) maintains a registry of *charset names* used on the Internet. Those charset names are very close in meaning to the Unicode character encoding model's concept of character encoding schemes, and all of the Unicode character encoding schemes are, in fact, registered as *charsets*. While the two concepts are quite close and the names used are identical, some important differences may arise in terms of the requirements for each, particularly when it comes to handling of the byte order mark. Exercise due caution when equating the two.

Examples. *Figure 2-12* illustrates the Unicode character encoding schemes, showing how each is derived from one of the encoding forms by serialization of bytes.

Figure 2-12. Unicode Encoding Schemes

A	Ω	語	𝍸	UTF-32BE
00 00 00 41	00 00 03 A9	00 00 8A 9E	00 01 03 84	

A	Ω	語	𝍸	UTF-32LE
41 00 00 00	A9 03 00 00	9E 8A 00 00	84 03 01 00	

A	Ω	語	𝍸	UTF-16BE
00 41	03 A9	8A 9E	D8 00 DF 84	

A	Ω	語	𝍸	UTF-16LE
41 00	A9 03	9E 8A	00 D8 84 DF	

A	Ω	語	𝍸	UTF-8
41	CE A9	E8 AA 9E	F0 90 8E 84	
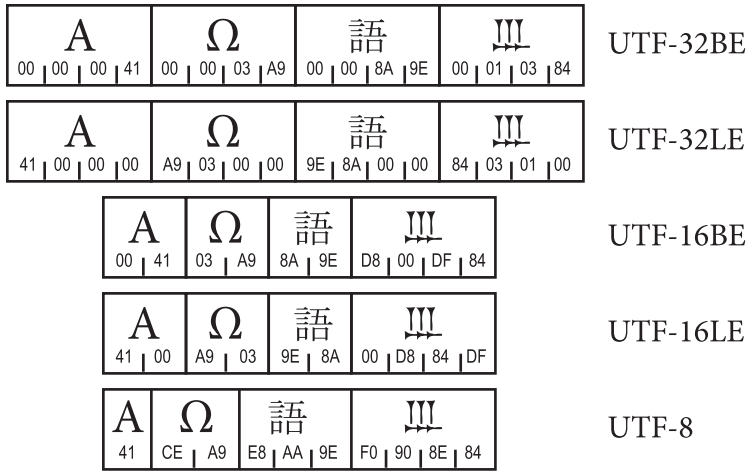
In *Figure 2-12*, the code units used to express each example character have been serialized into sequences of bytes. This figure should be compared with *Figure 2-11*, which shows the same characters before serialization into sequences of bytes. The "BE" lines show serialization in big-endian order, whereas the "LE" lines show the bytes reversed into little-endian order. For UTF-8, the code unit is just an 8-bit byte, so that there is no distinction between big-endian and little-endian order. UTF-32 and UTF-16 encoding schemes using the byte order mark are not shown in *Figure 2-12*, to keep the basic picture regarding serialization of bytes clearer.

For the detailed formal definition of the Unicode encoding schemes and conformance requirements, see *Section 3.10, Unicode Encoding Schemes*. For further general discussion about character encoding forms and character encoding schemes, both for the Unicode Standard and as applied to other character encoding standards, see Unicode Technical Report #17, "Unicode Character Encoding Model." For information about charsets and character conversion, see Unicode Technical Standard #22, "Character Mapping Markup Language (CharMapML)."

2.7 Unicode Strings

A Unicode string data type is simply an ordered sequence of code units. Thus a Unicode 8-bit string is an ordered sequence of 8-bit code units, a Unicode 16-bit string is an ordered sequence of 16-bit code units, and a Unicode 32-bit string is an ordered sequence of 32-bit code units.

Depending on the programming environment, a Unicode string may or may not be required to be in the corresponding Unicode encoding form. For example, strings in Java, C#, or ECMAScript are Unicode 16-bit strings, but are not necessarily well-formed UTF-16 sequences. In normal processing, it can be far more efficient to allow such strings to contain code unit sequences that are not well-formed UTF-16—that is, isolated surrogates. Because strings are such a fundamental component of every program, checking for isolated surrogates in every operation that modifies strings can create significant overhead, especially because supplementary characters are extremely rare as a percentage of overall text in programs worldwide.

It is straightforward to design basic string manipulation libraries that handle isolated surrogates in a consistent and straightforward manner. They cannot ever be interpreted as abstract characters, but they can be internally handled the same way as noncharacters where they occur. Typically they occur only ephemerally, such as in dealing with keyboard events. While an ideal protocol would allow keyboard events to contain complete strings, many allow only a single UTF-16 code unit per event. As a sequence of events is transmitted to the application, a string that is being built up by the application in response to those events may contain isolated surrogates at any particular point in time.

Whenever such strings are specified to be in a particular Unicode encoding form—even one with the same code unit size—the string must not violate the requirements of that encoding form. For example, isolated surrogates in a Unicode 16-bit string are not allowed when that string is specified to be *well-formed* UTF-16. A number of techniques are available for dealing with an isolated surrogate, such as omitting it, converting it into U+FFFD REPLACEMENT CHARACTER to produce well-formed UTF-16, or simply halting the processing of the string with an error. (See *Section 3.9, Unicode Encoding Forms*.)

2.8 Unicode Allocation

For convenience, the encoded characters of the Unicode Standard are grouped by linguistic and functional categories, such as script or writing system. For practical reasons, there are occasional departures from this general principle, as when punctuation associated with the ASCII standard is kept together with other ASCII characters in the range U+0020..U+007E rather than being grouped with other sets of general punctuation characters. By and large, however, the code charts are arranged so that related characters can be found near each other in the charts.

Grouping encoded characters by script or other functional categories offers the additional benefit of supporting various space-saving techniques in actual implementations, as for building tables or fonts.

For more information on writing systems, see *Section 6.1, Writing Systems*.

2.8.1 Planes

The Unicode codespace consists of the single range of numeric values from 0 to $10FFFF_{16}$, but in practice it has proven convenient to think of the codespace as divided up into *planes* of characters—each plane consisting of 64K code points. Because of these numeric conventions, the Basic Multilingual Plane is occasionally referred to as *Plane 0*. The last four hexadecimal digits in each code point indicate a character's position inside a plane. The remaining digits indicate the plane. For example, U+23456 CJK UNIFIED IDEOGRAPH-23456 is found at location 3456_{16} in Plane 2.

Basic Multilingual Plane. The Basic Multilingual Plane (BMP, or Plane 0) contains the common-use characters for all the modern scripts of the world as well as many historical and rare characters. By far the majority of all Unicode characters for almost all textual data can be found in the BMP.

Supplementary Multilingual Plane. The Supplementary Multilingual Plane (SMP, or Plane 1) is dedicated to the encoding of characters for scripts or symbols which either could not be fit into the BMP or see very infrequent usage. This includes many historic scripts, a number of lesser-used contemporary scripts, special-purpose invented scripts, notational systems or large pictographic symbol sets, and occasionally historic extensions of scripts whose core sets are encoded on the BMP.

Examples include Gothic (historic), Shavian (special-purpose invented), Musical Symbols (notational system), Domino Tiles (pictographic), and Ancient Greek Numbers (historic extension for Greek). A number of scripts, whether of historic and contemporary use, do not yet have their characters encoded in the Unicode Standard. The majority of scripts currently identified for encoding will eventually be allocated

in the SMP. As a result, some areas of the SMP will experience common, frequent usage.

Additional Ideographic Planes. The Supplementary Ideographic Plane (SIP, or Plane 2) and Tertiary Ideographic Plane (TIP, or Plane 3) are intended as an additional allocation areas for those CJK characters that could not be fit in the blocks set aside for more common CJK characters in the BMP. While there are a small number of common-use CJK characters in the SIP (for example, for Cantonese usage), the vast majority of Plane 2 characters are extremely rare or of historical interest only.

Supplementary Special-purpose Plane. The Supplementary Special-purpose Plane (SSP, or Plane 14) is the spillover allocation area for format control characters that do not fit into the small allocation areas for format control characters in the BMP.

Private Use Planes. The two Private Use Planes (Planes 15 and 16) are allocated for private use. Those two planes contain a total of 131,068 private-use characters to supplement the 6,400 private-use characters located in the BMP.

2.8.2 Allocation Areas and Blocks

Allocation Areas. The Unicode Standard does not have any normatively defined concept of *areas* or *zones* for the BMP (or other planes), but it is often handy to refer to the allocation areas of the BMP by the general types of the characters they include. These areas are merely a rough organizational device and do not restrict the types of characters that may end up being allocated in them. The description and ranges of areas may change from version to version of the standard as more new scripts, symbols, and other characters are encoded in previously reserved ranges.

Blocks. The various allocation areas are, in turn, divided up into character *blocks* (see D10b in *Section 3.4, Characters and Encoding*), which are normatively defined, and which are used to structure the actual code charts. For a complete listing of the normative blocks in the Unicode Standard, see Blocks.txt in the Unicode Character Database.

The normative status of blocks should not, however, be taken as indicating that they define significant sets of characters. For the most part, the blocks serve only as ranges to divide up the code charts and do not necessarily imply anything else about the types of characters found in the block. Block identity cannot be taken as a reliable guide to the source, use, or properties of characters, for example, and it cannot be reliably used alone to process characters. In particular:

- Blocks are simply named ranges, and many contain reserved code points.
- Characters used in a single writing system may be found in several different blocks. For example, characters used for letters for Latin-based writing systems are found in at least 14 different blocks: Basic Latin, Latin-1 Supplement, Latin

Extended-A, Latin Extended-B, Latin Extended-C, Latin Extended-D, Latin Extended-E, IPA Extensions, Phonetic Extensions, Phonetic Extensions Supplement, Latin Extended Additional, Spacing Modifier Letters, Combining Diacritical Marks, and Combining Diacritical Marks Supplement.

• Characters in a block may be used with different writing systems. For example, the *danda* character is encoded in the Devanagari block but is used with numerous other scripts; Arabic combining marks in the Arabic block are used with the Syriac script; and so on.

• Block definitions are not at all exclusive. For instance, many mathematical operator characters are not encoded in the Mathematical Operators block—and are not even in any block containing "Mathematical" in its name; many currency symbols are not found in the Currency Symbols block, and so on.

For reliable specification of the properties of characters, one should instead turn to the detailed, character-by-character property assignments available in the Unicode Character Database. See also *Chapter 4, Character Properties*. For further discussion of the relationship between the blocks in the Unicode standard and significant property assignments and sets of characters, see Unicode Standard Annex #24, "Unicode Script Property," and Unicode Technical Standard #18, "Unicode Regular Expressions."

Allocation Order. The allocation order of various scripts and other groups of characters reflects the historical evolution of the Unicode Standard. While there is a certain geographic sense to the ordering of the allocation areas for the scripts, this is only a very loose correlation.

Roadmap for Future Allocation. The unassigned ranges in the Unicode codespace will be filled with future script or symbol encodings on a space-available basis. The relevant character encoding committees follow an organized roadmap to help them decide where to encode new scripts and other characters within the available space. Until the characters are actually standardized, however, there are no absolute guarantees where future allocations will occur. In general, implementations should not make assumptions about where future scripts or other sets of symbols may be encoded based solely on the identity of neighboring blocks of characters already encoded.

See *Appendix B.3, Other Unicode Online Resources* for information about the roadmap and about the pipeline of approved characters in process for future publication.

2.8.3 Assignment of Code Points

Code points in the Unicode Standard are assigned using the following guidelines:

- Where there is a single accepted standard for a script, the Unicode Standard generally follows it for the relative order of characters within that script.

- The first 256 codes follow precisely the arrangement of ISO/IEC 8859-1 (Latin 1), of which 7-bit ASCII (ISO/IEC 646 IRV) accounts for the first 128 code positions.

- Characters with common characteristics are located together contiguously. For example, the primary Arabic character block was modeled after ISO/IEC 8859-6. The Arabic script characters used in Persian, Urdu, and other languages, but not included in ISO/IEC 8859-6, are allocated after the primary Arabic character block. Right-to-left scripts are grouped together.

- In most cases, scripts with fewer than 128 characters are allocated so as not to cross 128-code-point boundaries (that is, they fit in ranges nn00..nn7F or nn80..nnFF). For supplementary characters, an additional constraint not to cross 1,024-code-point boundaries is applied (that is, scripts fit in ranges nn000..nn3FF, nn400..nn7FF, nn800..nnBFF, or nnC00..nnFFF). Such constraints enable better optimizations for tasks such as building tables for access to character properties.

- Codes that represent letters, punctuation, symbols, and diacritics that are generally shared by multiple languages or scripts are grouped together in several locations.

- The Unicode Standard does not correlate character code allocation with language-dependent collation or case. For more information on collation order, see Unicode Technical Standard #10, "Unicode Collation Algorithm."

- Unified CJK ideographs are laid out in multiple blocks, each of which is arranged according to the Han ideograph arrangement defined in *Section 18.1, Han*. This ordering is roughly based on a radical-stroke count order.

2.9 Details of Allocation

This section provides a more detailed summary of the way characters are allocated in the Unicode Standard. *Figure 2-13* gives an overall picture of the allocation areas of the Unicode Standard, with an emphasis on the identities of the planes. The following subsections discuss the allocation details for specific planes.

Figure 2-13. Unicode Allocation
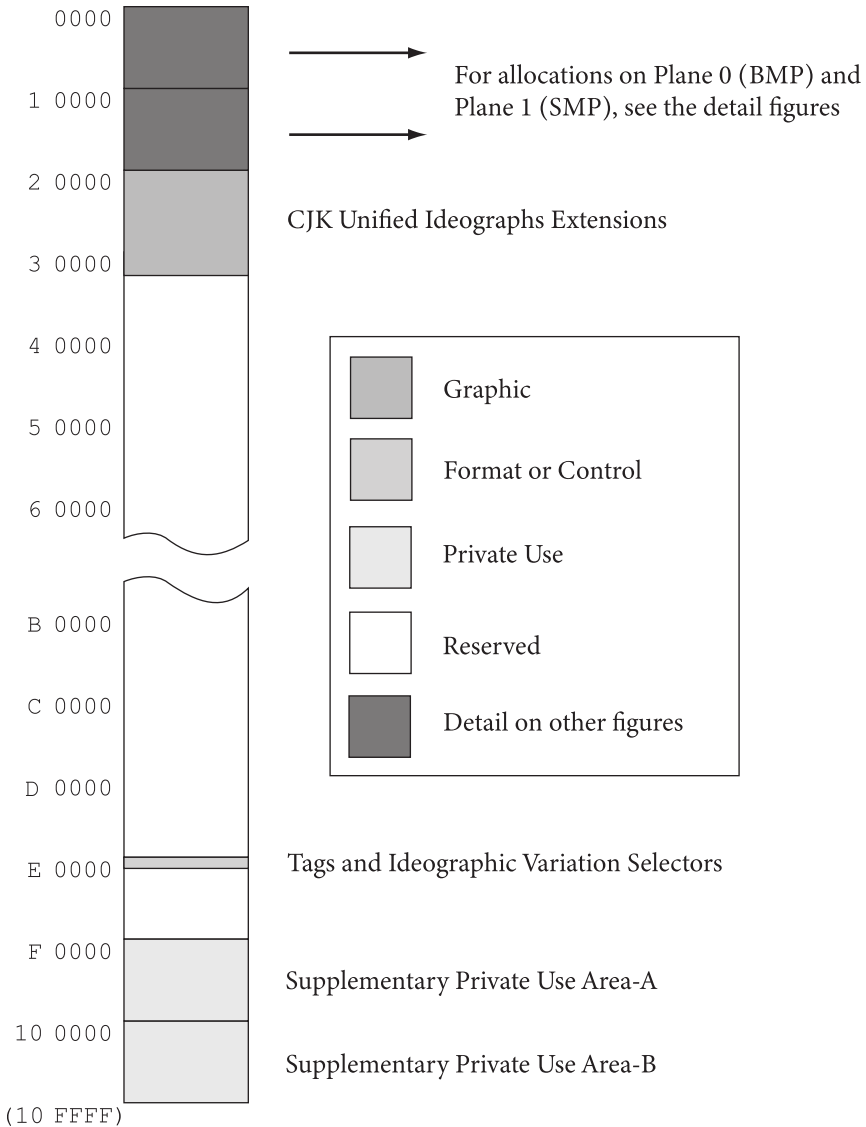
2.9.1 Plane 0 (BMP)

Figure 2-14 shows the Basic Multilingual Plane (BMP) in an expanded format to illustrate the allocation substructure of that plane in more detail. This section describes each allocation area, in the order of their location on the BMP.

Figure 2-14. Allocation on the BMP

0000	0000-00FF ASCII & Latin-1 Compatibility Area
	0100-058F General Scripts Area
	0590-08FF General Scripts Area (RTL)
0900	
	0900-1FFF General Scripts Area
2000	
	2000-2BFF Punctuation and Symbols Area
2C00	2C00-2DFF General Scripts Area
3400	2E00-2E7F Supplemental Punctuation Area
	2E80-33FF CJK Miscellaneous Area
	3400-9FFF CJKV Unified Ideographs Area (not to scale)
A000	A000-ABFF General Scripts Area (Asia & Africa)
AC00	
	AC00-D7FF Hangul Syllables Area
D800	D800-DFFF Surrogate Codes
E000	
	E000-F8FF Private Use Area
F900	
(FFFF)	F900-FFFF Compatibility and Specials Area

ASCII and Latin-1 Compatibility Area. For compatibility with the ASCII and ISO 8859-1, Latin-1 standards, this area contains the same repertoire and ordering as Latin-1. Accordingly, it contains the basic Latin alphabet, European digits, and then the same collection of miscellaneous punctuation, symbols, and additional Latin letters as are found in Latin-1.

General Scripts Area. The General Scripts Area contains a large number of modern-use scripts of the world, including Latin, Greek, Cyrillic, Arabic, and so on. Most of the characters encoded in this area are graphic characters. A subrange of the General Scripts Area is set aside for right-to-left scripts, including Hebrew, Arabic, Thaana, and N'Ko.

Punctuation and Symbols Area. This area is devoted mostly to all kinds of symbols, including many characters for use in mathematical notation. It also contains general punctuation, as well as most of the important format control characters.

Supplementary General Scripts Area. This area contains scripts or extensions to scripts that did not fit in the General Scripts Area itself. It contains the Glagolitic, Coptic, and Tifinagh scripts, plus extensions for the Latin, Cyrillic, Georgian, and Ethiopic scripts.

CJK Miscellaneous Area. The CJK Miscellaneous Area contains some East Asian scripts, such as Hiragana and Katakana for Japanese, punctuation typically used with East Asian scripts, lists of CJK radical symbols, and a large number of East Asian compatibility characters.

CJKV Ideographs Area. This area contains almost all the unified Han ideographs in the BMP. It is subdivided into a block for the Unified Repertoire and Ordering (the initial block of 20,902 unified Han ideographs plus a small number of later additions) and another block containing Extension A (an additional 6,582 unified Han ideographs).

General Scripts Area (Asia and Africa). This area contains numerous blocks for additional scripts of Asia and Africa, such as Yi, Cham, Vai, and Bamum. It also contains more spillover blocks with additional characters for the Latin, Devanagari, Myanmar, and Hangul scripts.

Hangul Area. This area consists of one large block containing 11,172 precomposed Hangul syllables, and one small block with additional, historic Hangul jamo extensions.

Surrogates Area. The Surrogates Area contains *only* surrogate code points and *no* encoded characters. See *Section 23.6, Surrogates Area*, for more details.

Private Use Area. The Private Use Area in the BMP contains 6,400 private-use characters.

Compatibility and Specials Area. This area contains many compatibility variants of characters from widely used corporate and national standards that have other representations in the Unicode Standard. For example, it contains Arabic presentation forms, whereas the basic characters for the Arabic script are located in the General Scripts Area. The Compatibility and Specials Area also contains twelve CJK unified ideographs, a few important format control characters, the basic variation selectors, and other special characters. See *Section 23.8, Specials*, for more details.

2.9.2 Plane 1 (SMP)

Figure 2-15 shows Plane 1, the Supplementary Multilingual Plane (SMP), in expanded format to illustrate the allocation substructure of that plane in more detail.

Figure 2-15. Allocation on Plane 1


```
1 0000    General Scripts Area
1 0800    General Scripts Area (RTL)
1 1000    General Scripts Area
1 2000

          Cuneiform & Hieroglyphic Area

1 6000    General Scripts Area
1 7000

          Ideographic Scripts Area

1 BC00    General Scripts Area
1 D000
          Symbols Area
1 E800    General Scripts Area (RTL)
1 F000    Symbols Area
(1 FFFF)
```
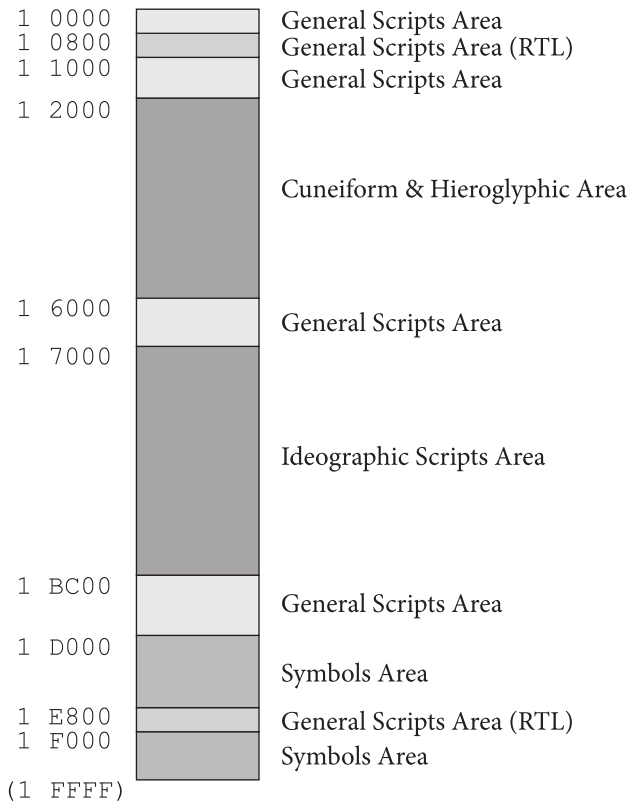
General Scripts Areas. These areas contain a large number of historic scripts, as well as a few regional scripts which have some current use. The first of these areas also contains a small number of symbols and numbers associated with ancient scripts.

General Scripts Areas (RTL). There are two subranges in the SMP which are set aside for historic right-to-left scripts, such as Phoenician, Kharoshthi, and Avestan. The second of these also defaults to Bidi_Class = R and is reserved for the encoding of other historic right-to-left scripts or symbols.

Cuneiform and Hieroglyphic Area. This area contains three large, ancient scripts: Sumero-Akkadian Cuneiform, Egyptian Hieroglyphs, and Anatolian Hieroglyphs. Other large hieroglyphic and pictographic scripts will be allocated in this area in the future.

Ideographic Scripts Area. This area is set aside for large, historic siniform (but non-Han) logosyllabic scripts such as Tangut, Jurchen, and Khitan, and other East Asian logosyllabic scripts such as Naxi. As of Unicode 12.0, this area contains a large set of Tangut ideographs and components, the Nüshu script, and a large set of hentaigana (historic, variant form kana) characters.

Symbols Areas. The first of these SMP Symbols Areas contains sets of symbols for notational systems, such as musical symbols, shorthands, and mathematical alphanumeric symbols. The second contains various game symbols, and large sets of miscellaneous symbols and pictographs, mostly used in compatibility mapping of East Asian character sets. Notable among these are emoji and emoticons.

2.9.3 Plane 2 (SIP)

Plane 2, the Supplementary Ideographic Plane (SIP), consists primarily of one big area, starting from the first code point in the plane, that is dedicated to encoding additional unified CJK characters. A much smaller area, toward the end of the plane, is dedicated to additional CJK compatibility ideographic characters—which are basically just duplicated character encodings required for round-trip conversion to various existing legacy East Asian character sets. The CJK compatibility ideographic characters in Plane 2 are currently all dedicated to round-trip conversion for the CNS standard and are intended to supplement the CJK compatibility ideographic characters in the BMP, a smaller number of characters dedicated to round-trip conversion for various Korean, Chinese, and Japanese standards.

2.9.4 Plane 3 (TIP)

Plane 3, the Tertiary Ideographic Plane (TIP), has a large area, starting from the first code point in the plane, that is dedicated to encoding additional unified CJK characters. This plane may also allocate blocks for historic precursors to the Han script—most notably the Seal Script.

2.9.5 Other Planes

The first 4,096 code positions on Plane 14 form an area set aside for special characters that have the Default_Ignorable_Code_Point property. A small number of tag characters, plus some supplementary variation selection characters, have been allocated there. All remaining code positions on Plane 14 are reserved for future allocation of other special-purpose characters.

Plane 15 and Plane 16 are allocated for private use. Those two planes contain a total of 131,068 private-use characters, to supplement the 6,400 private-use characters located in the BMP.

All other planes are reserved; there are no characters assigned in them. The last two code positions of *all* planes are permanently set aside as noncharacters. (See *Section 2.13, Special Characters*).

2.10 Writing Direction

Individual writing systems have different conventions for arranging characters into lines on a page or screen. Such conventions are referred to as a script's *directionality*. For example, in the Latin script, characters are arranged horizontally from left to right to form lines, and lines are arranged from top to bottom, as shown in the first example of *Figure 2-16*.

Figure 2-16. Writing Directions
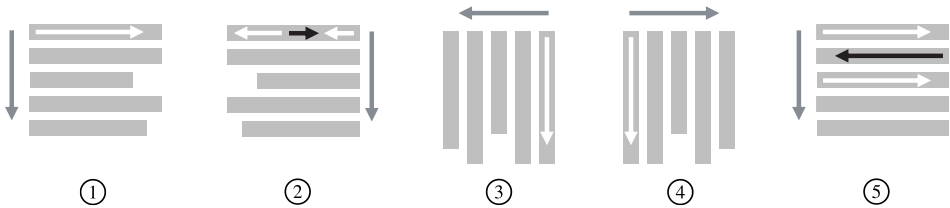
Bidirectional. In most Semitic scripts such as Hebrew and Arabic, characters are arranged from right to left into lines, although digits run the other way, making the scripts inherently bidirectional, as shown in the second example in *Figure 2-16*. In addition, left-to-right and right-to-left scripts are frequently used together. In all such cases, arranging characters into lines becomes more complex. The Unicode Standard defines an algorithm to determine the layout of a line, based on the inherent directionality of each character, and supplemented by a small set of directional controls. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," for more information.

Vertical. East Asian scripts are frequently written in vertical lines in which characters are arranged from top to bottom. Lines are arranged from right to left, as shown in the third example in *Figure 2-16*. Such scripts may also be written horizontally, from left to right. Most East Asian characters have the same shape and orientation when displayed horizontally or vertically, but many punctuation characters change their shape when displayed vertically. In a vertical context, letters and words from other scripts are generally rotated through 90-degree angles so that they, too, read from top to bottom. Unicode Technical Report #50, "Unicode Vertical Text Layout," defines a character property which is useful in determining the correct orientation of characters when laid out vertically in text.

In contrast to the bidirectional case, the choice to lay out text either vertically or horizontally is treated as a formatting style. Therefore, the Unicode Standard does not provide directionality controls to specify that choice.

Mongolian is usually written from top to bottom, with lines arranged from left to right, as shown in the fourth example. When Mongolian is written horizontally, the characters are rotated.

Boustrophedon. Early Greek used a system called *boustrophedon* (literally, "ox-turning"). In boustrophedon writing, characters are arranged into horizontal lines, but the individual lines alternate between right to left and left to right, the way an ox goes back and forth when plowing a field, as shown in the fifth example. The letter images are mirrored in accordance with the direction of each individual line.

Other Historical Directionalities. Other script directionalities are found in historical writing systems. For example, some ancient Numidian texts are written from bottom to top, and Egyptian hieroglyphics can be written with varying directions for individual lines.

The historical directionalities are of interest almost exclusively to scholars intent on reproducing the exact visual content of ancient texts. The Unicode Standard does not provide direct support for them. Fixed texts can, however, be written in boustrophedon or in other directional conventions by using hard line breaks and directionality overrides or the equivalent markup.

2.11 Combining Characters

Combining Characters. In concept, a combining character is a mark of some kind intended to be positioned relative to some other character, which is referred to as its associated base character. In the code charts, combining characters are depicted with an associated dotted circle, which stands in for the base. The visible mark for the combining character may appear above, below, to either side, crossing through, or even surrounding the dotted circle. When rendered, the glyphs that depict these characters are intended to be positioned relative to the glyph depicting the preceding base character in some combination.

The Unicode Standard distinguishes three types of combining characters: spacing, nonspacing, and enclosing. Nonspacing combining characters do not occupy a spacing position by themselves. Nevertheless, the combination of a base character and a nonspacing character may have a different advance width than the base character by itself. For example, an [î] may be slightly wider than a plain [i]. The detailed classification of which combining characters are spacing, nonspacing, or enclosing is provided in the Unicode Character Database.

All combining characters can be applied to any base character and can, in principle, be used with any script. As with other characters, the allocation of a combining character to one block or another identifies only its primary usage; it is not intended to define or limit the range of characters to which it may be applied. *In the Unicode Standard, all sequences of character codes are permitted.*

This does not create an obligation on implementations to support all possible combinations equally well. Thus, while application of an Arabic annotation mark to a Han character or a Devanagari consonant is permitted, it is unlikely to be supported well in rendering or to make much sense.

Diacritics. Diacritics are the principal class of nonspacing combining characters used with the Latin, Greek, and Cyrillic scripts and their relatives. In the Unicode Standard, the term "diacritic" is defined very broadly to include accents as well as other nonspacing marks.

Symbol Diacritics. Some diacritical marks are applied primarily to symbols. These combining marks are allocated in the Combining Diacritical Marks for Symbols block, to distinguish them from diacritical marks applied primarily to letters.

Enclosing Combining Marks. *Figure 2-17* shows examples of combining enclosing marks for symbols. The combination of an enclosing mark with a base character has the appearance of a symbol. As discussed in "Properties" later in this section, it is best to limit the use of combining enclosing marks to characters that represent

symbols. This limitation minimizes the potential for surprises resulting from mismatched character properties.

A few symbol characters are intended primarily for use with enclosing combining marks. For example, U+2621 CAUTION SIGN is a winding road symbol that can be used in combination with U+20E4 COMBINING ENCLOSING UPWARD POINTING TRIANGLE or U+20DF COMBINING ENCLOSING DIAMOND. However, the enclosing combining marks can also be used in combination with arbitrary symbols, as illustrated by applying U+20E0 COMBINING ENCLOSING CIRCLE BACKSLASH to U+2615 HOT BEVERAGE to create a "no drinks allowed" symbol. Furthermore, no formal restriction prevents enclosing combining marks from being used with non-symbols, as illustrated by applying U+20DD COMBINING ENCLOSING CIRCLE to U+062D ARABIC LETTER HAH to represent a circled *hah*.

Figure 2-17. Combining Enclosing Marks for Symbols
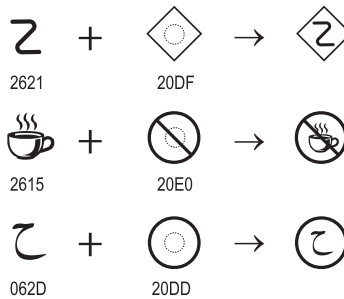
Script-Specific Combining Characters. Some scripts, such as Hebrew, Arabic, and the scripts of India and Southeast Asia, have both spacing and nonspacing combining characters specific to those scripts. Many of these combining characters encode vowel letters. As such, they are not generally referred to as diacritics, but may have script-specific terminology such as *harakat* (Arabic) or *matra* (Devanagari). See *Section 7.9, Combining Marks*.

2.11.1 Sequence of Base Characters and Combining Characters

In the Unicode Standard, all combining characters are to be used in sequence following the base characters to which they apply. The sequence of Unicode characters <U+0061 "a" LATIN SMALL LETTER A, U+0308 "◌̈" COMBINING DIAERESIS, U+0075 "u" LATIN SMALL LETTER U> unambiguously represents "äu" and not "aü", as shown in *Figure 2-18*.

Figure 2-18. Sequence of Base Characters and Combining Characters

$$a + \ddot{\circ} + u \quad \rightarrow \quad \ddot{a}u \; (\textit{not } a\ddot{u})$$
<center>0061   0308   0075</center>

Ordering. The ordering convention used by the Unicode Standard—placing combining marks after the base character to which they apply—is consistent with the logical order of combining characters in Semitic and Indic scripts, the great majority of which (logically or phonetically) follow the base characters with which they are associated. This convention also conforms to the way modern font technology handles the rendering of nonspacing graphical forms (glyphs), so that mapping from character memory representation order to font rendering order is simplified. It is different from the convention used in the bibliographic standard ISO 5426.

Indic Vowel Signs. Some Indic vowel signs are rendered to the left of a consonant letter or consonant cluster, even though their logical order in the Unicode encoding follows the consonant letter. In the charts, these vowels are depicted to the left of dotted circles (see *Figure 2-19*). The coding of these vowels in pronunciation order and not in visual order is consistent with the ISCII standard.

Figure 2-19. Reordered Indic Vowel Signs

$$फ + ि\circ \rightarrow फि$$
<center>092B   093F</center>

Properties. A sequence of a base character plus one or more combining characters generally has the same properties as the base character. For example, "A" followed by "ˆ" has the same properties as "Â". For this reason, most Unicode algorithms ensure that such sequences behave the same way as the corresponding base character. However, when the combining character is an enclosing combining mark—in other words, when its General_Category value is Me—the resulting sequence has the appearance of a symbol. In *Figure 2-20*, enclosing the *exclamation mark* with U+20E4 COMBINING ENCLOSING UPWARD POINTING TRIANGLE produces a sequence that looks like U+26A0 WARNING SIGN.

Figure 2-20. Properties and Combining Character Sequences

$$! + \triangle \rightarrow \triangle\!\!\! \neq \triangle\!\!\!$$
<center>0021   20E4   26A0</center>

Because the properties of U+0021 EXCLAMATION MARK are that of a punctuation character, they are different from those of U+26A0 WARNING SIGN. For example, the two will behave differently for line breaking. To avoid unexpected results, it is best to limit the use of combining enclosing marks to characters that encode symbols. For

that reason, the *warning sign* is separately encoded as a miscellaneous symbol in the Unicode Standard and does not have a decomposition.

2.11.2 Multiple Combining Characters

In some instances, more than one combining character is applied to a single base character (see *Figure 2-21*). The Unicode Standard does not restrict the number of combining characters that may follow a base character. The following discussion summarizes the default treatment of multiple combining characters. (For further discussion, see *Section 3.6, Combination*.)

Figure 2-21. Multiple Combining Characters
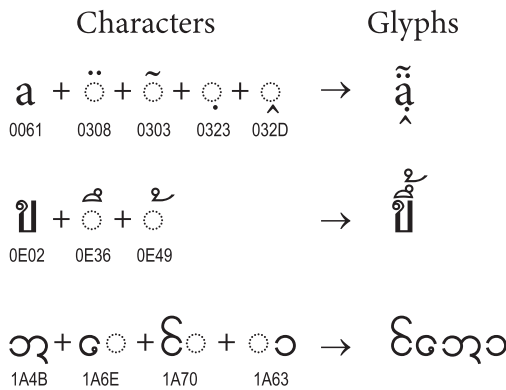
If the combining characters can interact typographically—for example, U+0304 COMBINING MACRON and U+0308 COMBINING DIAERESIS—then the order of graphic display is determined by the order of coded characters (see *Table 2-5*). By default, the diacritics or other combining characters are positioned from the base character's glyph outward. Combining characters placed above a base character will be stacked vertically, starting with the first encountered in the logical store and continuing for as many marks above as are required by the character codes following the base character. For combining characters placed below a base character, the situation is reversed, with the combining characters starting from the base character and stacking downward. For combining characters placed to the left of a base character, the combining characters start from the base character and are arranged leftward. For combining characters placed to the right of a base character, the combining characters start from the base character and are arranged rightward.

When combining characters do not interact typographically, the relative ordering of contiguous combining marks cannot result in any visual distinction and thus is insignificant.

Table 2-5. Interaction of Combining Characters

Glyph	Equivalent Sequences
ã	LATIN SMALL LETTER A WITH TILDE
	LATIN SMALL LETTER A + COMBINING TILDE
ȧ	LATIN SMALL LETTER A WITH DOT ABOVE
	LATIN SMALL LETTER A + COMBINING DOT ABOVE
ẫ	LATIN SMALL LETTER A WITH TILDE + COMBINING DOT BELOW
	LATIN SMALL LETTER A + COMBINING TILDE + COMBINING DOT BELOW
	LATIN SMALL LETTER A WITH DOT BELOW + COMBINING TILDE
	LATIN SMALL LETTER A + COMBINING DOT BELOW + COMBINING TILDE
ạ̇	LATIN SMALL LETTER A WITH DOT BELOW + COMBINING DOT ABOVE
	LATIN SMALL LETTER A + COMBINING DOT BELOW + COMBINING DOT ABOVE
	LATIN SMALL LETTER A WITH DOT ABOVE + COMBINING DOT BELOW
	LATIN SMALL LETTER A + COMBINING DOT ABOVE + COMBINING DOT BELOW
ấ	LATIN SMALL LETTER A WITH CIRCUMFLEX AND ACUTE
	LATIN SMALL LETTER A WITH CIRCUMFLEX + COMBINING ACUTE ACCENT
	LATIN SMALL LETTER A + COMBINING CIRCUMFLEX ACCENT + COMBINING ACUTE ACCENT
á̂	LATIN SMALL LETTER A WITH ACUTE + COMBINING CIRCUMFLEX ACCENT
	LATIN SMALL LETTER A + COMBINING ACUTE ACCENT + COMBINING CIRCUMFLEX ACCENT

Another example of multiple combining characters above the base character can be found in Thai, where a consonant letter can have above it one of the vowels U+0E34 through U+0E37 and, above that, one of four tone marks U+0E48 through U+0E4B. The order of character codes that produces this graphic display is *base consonant character + vowel character + tone mark character*, as shown in *Figure 2-21*.

Many combining characters have specific typographical traditions that provide detailed rules for the expected rendering. These rules override the default stacking behavior. For example, certain combinations of combining marks are sometimes positioned horizontally rather than stacking or by ligature with an adjacent nonspacing mark (see *Table 2-6*). When positioned horizontally, the order of codes is reflected by positioning in the predominant direction of the script with which the codes are used. For example, in a left-to-right script, horizontal accents would be coded from left to right. In *Table 2-6*, the top example is correct and the bottom example is incorrect.

Such override behavior is associated with specific scripts or alphabets. For example, when used with the Greek script, the "breathing marks" U+0313 COMBINING COMMA ABOVE (*psili*) and U+0314 COMBINING REVERSED COMMA ABOVE (*dasia*) require that, when used together with a following acute or grave accent, they be

rendered side-by-side rather than the accent marks being stacked above the breathing marks. The order of codes here is *base character code + breathing mark code + accent mark code*. This example demonstrates the script-dependent or writing-system-dependent nature of rendering combining diacritical marks.

Table 2-6. Nondefault Stacking

Glyph	Sequence	Note
ἄ	GREEK SMALL LETTER ALPHA + COMBINING COMMA ABOVE (*psili*) + COMBINING ACUTE ACCENT (*oxia*)	This is correct
ἄ	GREEK SMALL LETTER ALPHA + COMBINING ACUTE ACCENT (*oxia*) + COMBINING COMMA ABOVE (*psili*)	This is incorrect

For additional examples of script-specific departure from default stacking of sequences of combining marks, see the discussion about the positioning of multiple points and marks in *Section 9.1, Hebrew*, the discussion of nondefault placement of Arabic vowel marks accompanying *Figure 9-5* in *Section 9.2, Arabic*, or the discussion of horizontal combination of titlo letters in Old Church Slavonic accompanying *Figure 7-6* in *Section 7.4, Cyrillic*.

Vietnamese is written with the Latin script, and regularly uses multiple diacritics on single letters when displaying tone marks above or below vowel letters that already contain diacritic marks. Acute or grave tone marks are often seen offset horizontally to the right or left of a circumflex mark above a vowel letter. These conventions are shown, for example, in the code chart font used for the precomposed Vietnamese vowel plus tone mark combinations in the code charts in the range U+1EA0..U+1EF9 in the Latin Extended Additional block. However, the exact placement of tone marks in Vietnamese text is considered stylistic, and often varies considerably from font to font.

For other types of nondefault stacking behavior, see the discussion about the positioning of combining parentheses in the subsection "Combining Diacritical Marks Extended: U+1AB0–U+1AFF" in *Section 7.9, Combining Marks*.

The Unicode Standard specifies default stacking behavior to offer guidance about which character codes are to be used in which order to represent the text, so that texts containing multiple combining marks can be interchanged reliably. The Unicode Standard does not aim to regulate or restrict typographical tradition. Fonts and rendering systems should be designed to override default stacking of multiple combining marks where necessary to produce the correct appearance for the relevant script and language context.

2.11.3 Ligated Multiple Base Characters

When the glyphs representing two base characters merge to form a ligature, the combining characters must be rendered correctly in relation to the ligated glyph (see *Figure 2-22*). Internally, the software must distinguish between the nonspacing marks that apply to positions relative to the first part of the ligature glyph and those that apply to the second part. (For a discussion of general methods of positioning nonspacing marks, see *Section 5.12, Strategies for Handling Nonspacing Marks*.)

Figure 2-22. Ligated Multiple Base Characters

$$f + \tilde{\circ} + i + \underset{.}{\circ} \rightarrow \tilde{f\!i}$$

0066   0303   0069   0323

For more information, see "Application of Combining Marks" in *Section 3.6, Combination*.

Ligated base characters with multiple combining marks do not commonly occur in most scripts. However, in some scripts, such as Arabic, this situation occurs quite often when vowel marks are used. It arises because of the large number of ligatures in Arabic, where each element of a ligature is a consonant, which in turn can have a vowel mark attached to it. Ligatures can even occur with three or more characters merging; vowel marks may be attached to each part.

2.11.4 Exhibiting Nonspacing Marks in Isolation

Nonspacing combining marks used by the Unicode Standard may be exhibited in apparent isolation by applying them to U+00A0 NO-BREAK SPACE. This convention might be employed, for example, when talking about the combining mark itself as a mark, rather than using it in its normal way in text (that is, applied as an accent to a base letter or in other combinations).

Prior to Version 4.1 of the Unicode Standard, the standard recommended the use of U+0020 SPACE for display of isolated combining marks. This practice is no longer recommended because of potential conflicts with the handling of sequences of U+0020 SPACE characters in such contexts as XML. For additional ways of displaying some diacritical marks, see "Spacing Clones of Diacritical Marks" in *Section 7.9, Combining Marks*.

2.11.5 "Characters" and Grapheme Clusters

End users have various concepts about what constitutes a letter or "character" in the writing system for their language or languages. The precise scope of these end-user "characters" depends on the particular written language and the orthography it uses.

In addition to the many instances of accented letters, they may extend to digraphs such as Slovak "ch", trigraphs or longer combinations, and sequences using spacing letter modifiers, such as "k^W". Such concepts are often important for processes such as collation, for the definition of characters in regular expressions, and for counting "character" positions within text. In instances such as these, what the user thinks of as a character may affect how the collation or regular expression will be defined or how the "characters" will be counted. Words and other higher-level text elements generally do not split within elements that a user thinks of as a character, even when the Unicode representation of them may consist of a sequence of encoded characters.

The variety of these end-user-perceived characters is quite great—particularly for digraphs, ligatures, or syllabic units. Furthermore, it depends on the particular language and writing system that may be involved. Despite this variety, however, the core concept "characters that should be kept together" can be defined for the Unicode Standard in a language-independent way. This core concept is known as a *grapheme cluster*. A grapheme cluster consists of a base character followed by any number of continuing characters, where a continuing character may include any nonspacing combining mark, certain spacing combining marks, or a join control.

An implementation operating on such a cluster would almost never want to break between its elements for rendering, editing, or other such text processes; the grapheme cluster is treated as a single unit. Unicode Standard Annex #29, "Unicode Text Segmentation," provides a complete formal definition of a grapheme cluster and discusses its application in the context of editing and other text processes. Implementations also may tailor the definition of a grapheme cluster, so that under limited circumstances, particular to one written language or another, the grapheme cluster may more closely pertain to what end users think of as "characters" for that language.

2.12 Equivalent Sequences

In cases involving two or more sequences considered to be equivalent, the Unicode Standard does not prescribe one particular sequence as being the *correct* one; instead, each sequence is merely equivalent to the others. *Figure 2-23* illustrates the two major forms of equivalent sequences formally defined by the Unicode Standard. In the first example, the sequences are canonically equivalent. Both sequences should display and be interpreted the same way. The second and third examples illustrate different compatibility sequences. Compatible-equivalent sequences may have format differences in display and may be interpreted differently in some contexts.

Figure 2-23. Equivalent Sequences

If an application or user attempts to distinguish between *canonically* equivalent sequences, as shown in the first example in *Figure 2-23*, there is no guarantee that other applications would recognize the same distinctions. To prevent the introduction of interoperability problems between applications, such distinctions must be avoided wherever possible. Making distinctions between compatibly equivalent sequences is less problematical. However, in restricted contexts, such as the use of identifiers, avoiding compatibly equivalent sequences reduces possible security issues. See Unicode Technical Report #36, "Unicode Security Considerations."

2.12.1 Normalization

Where a unique representation is required, a normalized form of Unicode text can be used to eliminate unwanted distinctions. The Unicode Standard defines four normalization forms: Normalization Form D (NFD), Normalization Form KD (NFKD), Normalization Form C (NFC), and Normalization Form KC (NFKC). Roughly speaking, NFD and NFKD decompose characters where possible, while NFC and NFKC compose characters where possible. For more information, see Unicode Standard Annex #15, "Unicode Normalization Forms," and *Section 3.11, Normalization Forms*.

A key part of normalization is to provide a unique canonical order for visually nondistinct sequences of combining characters. *Figure 2-24* shows the effect of canonical ordering for multiple combining marks applied to the same base character.

Figure 2-24. Canonical Ordering
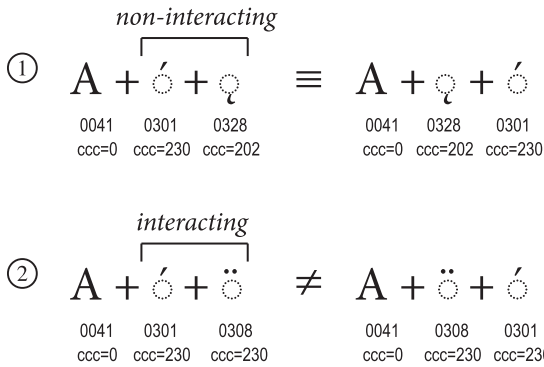
In the first row of *Figure 2-24*, the two sequences are visually nondistinct and, therefore, equivalent. The sequence on the right has been put into canonical order by reordering in ascending order of the Canonical_Combining_Class (ccc) values. The ccc values are shown below each character. The second row of *Figure 2-24* shows an example where combining marks interact typographically—the two sequences have different stacking order, and the order of combining marks is significant. Because the two combining marks have been given the same combining class, their ordering is retained under canonical reordering. Thus the two sequences in the second row are not equivalent.

2.12.2 Decompositions

Precomposed characters are formally known as decomposables, because they have decompositions to one or more *other* characters. There are two types of decompositions:

- *Canonical.* The character and its decomposition should be treated as essentially equivalent.
- *Compatibility.* The decomposition may remove some information (typically formatting information) that is important to preserve in particular contexts.

Types of Decomposables. Conceptually, a decomposition implies reducing a character to an equivalent sequence of constituent parts, such as mapping an accented character to a base character followed by a combining accent. The vast majority of nontrivial decompositions are indeed a mapping from a character code to a character sequence. However, in a small number of exceptional cases, there is a mapping from one character to another character, such as the mapping from *ohm* to *capital omega*.

Finally, there are the "trivial" decompositions, which are simply a mapping of a character to itself. They are really an indication that a character cannot be decomposed, but are defined so that all characters formally have a decomposition. The definition of *decomposable* is written to encompass only the nontrivial types of decompositions; therefore these characters are considered *nondecomposable*.

In summary, three types of characters are distinguished based on their decomposition behavior:

- *Canonical decomposable.* A character that is not identical to its canonical decomposition.

- *Compatibility decomposable.* A character whose compatibility decomposition is not identical to its canonical decomposition.

- *Nondecomposable.* A character that is identical to both its canonical decomposition and its compatibility decomposition. In other words, the character has trivial decompositions (decompositions to itself). Loosely speaking, these characters are said to have "no decomposition," even though, for completeness, the algorithm that defines decomposition maps such characters to themselves.

Because of the way decompositions are defined, a character cannot have a nontrivial canonical decomposition while having a trivial compatibility decomposition. Characters with a trivial compatibility decomposition are therefore always nondecomposables.

Examples. <u>Figure 2-25</u> illustrates these three types. Compatibility decompositions that are redundant because they are identical to the canonical decompositions are not shown.

Figure 2-25. Types of Decomposables

Nondecomposables

a	ø	Đ	ڇ
0061	00F8	0110	0681

Canonical decomposables	Compatibility decomposables

The figure illustrates two important points:

- Decompositions may be to single characters *or* to sequences of characters. Decompositions to a single character, also known as *singleton decompositions*, are seen for the *ohm sign* and the *halfwidth katakana ka* in <u>Figure 2-25</u>. Because of examples like these, decomposable characters in Unicode do not always consist of obvious, separate parts; one can know their status only by examining the data tables for the standard.

- A very small number of characters are both canonical and compatibility decomposable. The example shown in <u>Figure 2-25</u> is for the Greek hooked upsilon symbol with an acute accent. It has a canonical decomposition to one sequence and a compatibility decomposition to a different sequence.

For more precise definitions of these terms, see <u>Chapter 3, Conformance</u>.

2.12.3 Non-decomposition of Certain Diacritics

Most characters that one thinks of as being a letter "plus accent" have formal decompositions in the Unicode Standard. For example, see the canonical decomposable U+00C1 LATIN CAPITAL LETTER A WITH ACUTE shown in <u>Figure 2-25</u>. There are, however, exceptions involving certain types of diacritics and other marks.

Overlaid and Attached Diacritics. Based on the pattern for accented letters, implementers often also expect to encounter formal decompositions for characters which use various overlaid diacritics such as slashes and bars to form new Latin (or Cyrillic) letters. For example, one might expect a decomposition for U+00D8 LATIN CAPITAL LETTER O WITH STROKE involving U+0338 COMBINING LONG SOLIDUS OVERLAY.

However, such decompositions involving overlaid diacritics are not formally defined in the Unicode Standard.

For historical and implementation reasons, there are no decompositions for characters with overlaid diacritics such as slashes and bars, nor for most diacritic hooks, swashes, tails, and other similar modifications to the graphic form of a base character. In such cases, the generic identification of the overlaid element is not specific enough to identify which part of the base glyph is to be overlaid. The characters involved include prototypical overlaid diacritic letters as U+0268 LATIN SMALL LETTER I WITH STROKE, but also characters with hooks and descenders, such as U+0188 LATIN SMALL LETTER C WITH HOOK, U+049B CYRILLIC SMALL LETTER KA WITH DESCENDER, and U+0499 CYRILLIC SMALL LETTER ZE WITH DESCENDER.

There are three exceptional attached diacritics which are regularly decomposed, namely U+0327 COMBINING CEDILLA, U+0328 COMBINING OGONEK, and U+031B COMBINING HORN (which is used in Vietnamese letters).

Other Diacritics. There are other characters for which the name and glyph appear to imply the presence of a decomposable diacritic, but which have no decomposition defined in the Unicode Standard. A prominent example is the Pashto letter U+0681 ARABIC LETTER HAH WITH HAMZA ABOVE. In these cases, as for the overlaid diacritics, the composed character and the sequence of base letter plus combining diacritic are not equivalent, although their renderings would be very similar. See the text on "Combining Hamza Above" in *Section 9.2, Arabic* for further complications related to this and similar characters.

Character Names and Decomposition. One *cannot* determine the decomposition status of a Latin letter from its Unicode name, despite the existence of phrases such as "...WITH ACUTE" or "...WITH STROKE". The normative decomposition mappings listed in the Unicode Character Database are the only formal definition of decomposition status.

Simulated Decomposition in Processing. Because the Unicode characters with overlaid diacritics or similar modifications to their base form shapes have no formal decompositions, some kinds of text processing that would ordinarily use Normalization Form D (NFD) internally to separate base letters from accents may end up simulating decompositions instead. Effectively, this processing treats overlaid diacritics *as if* they were represented by a separately encoded combining mark. For example, a common operation in searching or matching is to sort (or match) while ignoring accents and diacritics on letters. This is easy to do with characters that formally decompose; the text is decomposed, and then the combining marks for the accents are ignored. However, for letters with overlaid diacritics, the effect of ignoring the diacritic has to be simulated instead with data tables that go beyond simple use of Unicode decomposition mappings.

Security Issue. The lack of formal decompositions for characters with overlaid diacritics means that there are increased opportunities for spoofing involving such characters. The display of a base letter plus a combining overlaid mark such as U+0335 COMBINING SHORT STROKE OVERLAY may look the same as the encoded base letter with bar diacritic, but the two sequences are not canonically equivalent and would not be folded together by Unicode normalization.

Implementations of writing systems which make use of letters with overlaid diacritics typically do not mix atomic representation (use of a precomposed letter with overlaid diacritic) with sequential representation (use of a sequence of base letter plus combining mark for the overlaid diacritic). Mixing these conventions is avoided precisely because the atomic representation and the sequential representation are not canonically equivalent. In most cases the atomic representation is the preferred choice, because of its convenience and more reliable display.

Security protocols for identifiers may disallow either the sequential representation or the atomic representation of a letter with an overlaid diacritic to try to minimize spoofing opportunities. However, when this is done, it is incumbent on the protocol designers first to verify whether the atomic or the sequential representation is in actual use. Disallowing the preferred convention, while instead forcing use of the unpreferred one for a particular writing system can have the unintended consequence of *increasing* confusion about use—and may thereby reduce the usability of the protocol for its intended purpose.

For more information and data for handling these confusable sequences involving overlaid diacritics, see Unicode Technical Report #36, "Unicode Security Considerations."

2.13 Special Characters

The Unicode Standard includes a small number of important characters with special behavior; some of them are introduced in this section. It is important that implementations treat these characters properly. For a list of these and similar characters, see *Section 4.12, Characters with Unusual Properties*; for more information about such characters, see *Section 23.1, Control Codes*; *Section 23.2, Layout Controls*; *Section 23.7, Noncharacters*; and *Section 23.8, Specials*.

2.13.1 Special Noncharacter Code Points

The Unicode Standard contains a number of code points that are intentionally *not* used to represent assigned characters. These code points are known as *noncharacters*. They are permanently reserved for internal use and are not used for open interchange of Unicode text. For more information on noncharacters, see *Section 23.7, Noncharacters*.

2.13.2 Byte Order Mark (BOM)

The UTF-16 and UTF-32 encoding forms of Unicode plain text are sensitive to the byte ordering that is used when serializing text into a sequence of bytes, such as when writing data to a file or transferring data across a network. Some processors place the least significant byte in the initial position; others place the most significant byte in the initial position. Ideally, all implementations of the Unicode Standard would follow only one set of byte order rules, but this scheme would force one class of processors to swap the byte order on reading and writing plain text files, even when the file never leaves the system on which it was created.

To have an efficient way to indicate which byte order is used in a text, the Unicode Standard contains two code points, U+FEFF ZERO WIDTH NO-BREAK SPACE (*byte order mark*) and U+FFFE (a noncharacter), which are the byte-ordered mirror images of each other. When a BOM is received with the opposite byte order, it will be recognized as a noncharacter and can therefore be used to detect the intended byte order of the text. The BOM is not a control character that selects the byte order of the text; rather, its function is to allow recipients to determine which byte ordering is used in a file.

Unicode Signature. An initial BOM may also serve as an implicit marker to identify a file as containing Unicode text. For UTF-16, the sequence FE_{16} FF_{16} (or its byte-reversed counterpart, FF_{16} FE_{16}) is exceedingly rare at the outset of text files that use other character encodings. The corresponding UTF-8 BOM sequence, EF_{16} BB_{16} BF_{16}, is also exceedingly rare. In either case, it is therefore unlikely to be confused with real text data. The same is true for both single-byte and multibyte encodings.

Data streams (or files) that begin with the U+FEFF byte order mark are likely to contain Unicode characters.

Conformance to the Unicode Standard does not require the use of the BOM as such a signature. See *Section 23.8, Specials*, for more information on the byte order mark and its use as an encoding signature.

2.13.3 Layout and Format Control Characters

The Unicode Standard defines several characters that are used to control joining behavior, bidirectional ordering control, and alternative formats for display. Their specific use in layout and formatting is described in *Section 23.2, Layout Controls*.

2.13.4 The Replacement Character

U+FFFD REPLACEMENT CHARACTER is the general substitute character in the Unicode Standard. It can be substituted for any "unknown" character in another encoding that cannot be mapped in terms of known Unicode characters (see *Section 5.3, Unknown and Missing Characters*, and *Section 23.8, Specials*).

2.13.5 Control Codes

In addition to the special characters defined in the Unicode Standard for a number of purposes, the standard incorporates the legacy control codes for compatibility with the ISO/IEC 2022 framework, ASCII, and the various protocols that make use of control codes. Rather than simply being defined as byte values, however, the legacy control codes are assigned to Unicode code points: U+0000..U+001F, U+007F..U+009F. Those code points for control codes must be represented consistently with the various Unicode encoding forms when they are used with other Unicode characters. For more information on control codes, see *Section 23.1, Control Codes*.

2.14 Conforming to the Unicode Standard

Conformance requirements are a set of unambiguous criteria to which a conformant implementation of a standard must adhere, so that it can interoperate with other conformant implementations. The universal scope of the Unicode Standard complicates the task of rigorously defining such conformance requirements for all aspects of the standard. Making conformance requirements overly confining runs the risk of unnecessarily restricting the breadth of text operations that can be implemented with the Unicode Standard or of limiting them to a one-size-fits-all lowest common denominator. In many cases, therefore, the conformance requirements deliberately cover only minimal requirements, falling far short of providing a complete description of the behavior of an implementation. Nevertheless, there are many core aspects of the standard for which a precise and exhaustive definition of conformant behavior is possible.

This section gives examples of both conformant and nonconformant implementation behavior, illustrating key aspects of the formal statement of conformance requirements found in *Chapter 3, Conformance*.

2.14.1 Characteristics of Conformant Implementations

An implementation that conforms to the Unicode Standard has the following characteristics:

It treats characters according to the specified Unicode encoding form.

- The byte sequence <20 20> is interpreted as U+2020 '†' DAGGER in the UTF-16 encoding form.
- The same byte sequence <20 20> is interpreted as the sequence of two spaces, <U+0020, U+0020>, in the UTF-8 encoding form.

It interprets characters according to the identities, properties, and rules defined for them in this standard.

- U+2423 is '␣' OPEN BOX, *not* 'ぃ' *hiragana small i* (which is the meaning of the bytes 2423_{16} in JIS).
- U+00F4 'ô' is equivalent to U+006F 'o' followed by U+0302 'ˆ', but *not equivalent to* U+0302 followed by U+006F.
- U+05D0 'א' followed by U+05D1 'ב' looks like 'אב', *not* 'בא' when displayed.

When an implementation supports the display of Arabic, Hebrew, or other right-to-left characters and displays those characters, they must be ordered according to the

Bidirectional Algorithm described in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

When an implementation supports Arabic, Devanagari, or other scripts with complex shaping for their characters and displays those characters, at a minimum the characters are shaped according to the relevant block descriptions. (More sophisticated shaping can be used if available.)

2.14.2 Unacceptable Behavior

It is unacceptable for a conforming implementation:

To use unassigned codes.

- U+2073 is unassigned and not usable for '³' (*superscript 3*) or any other character.

To corrupt unsupported characters.

- U+03A1 "P" GREEK CAPITAL LETTER RHO should not be changed to U+00A1 (first byte dropped), U+0050 (mapped to Latin letter *P*), U+A103 (bytes reversed), or anything other than U+03A1.

To remove or alter uninterpreted code points in text that purports to be unmodified.

- U+2029 is PARAGRAPH SEPARATOR and should not be dropped by applications that do not support it.

2.14.3 Acceptable Behavior

It is acceptable for a conforming implementation:

To support only a subset of the Unicode characters.

- An application might not provide mathematical symbols or the Thai script, for example.

To transform data knowingly.

- Uppercase conversion: 'a' transformed to 'A'

- Romaji to kana: 'kyo' transformed to きょ

- Decomposition: U+247D '(10)' decomposed to <U+0028, U+0031, U+0030, U+0029>

To build higher-level protocols on the character set.

- Examples are defining a file format for compression of characters or for use with rich text.

To define private-use characters.

- Examples of characters that might be defined for private use include additional ideographic characters (*gaiji*) or existing corporate logo characters.

To not support the Bidirectional Algorithm or character shaping in implementations that do not support complex scripts, such as Arabic and Devanagari.

To not support the Bidirectional Algorithm or character shaping in implementations that do not display characters, as, for example, on servers or in programs that simply parse or transcode text, such as an XML parser.

Code conversion between other character encodings and the Unicode Standard will be considered conformant if the conversion is accurate in both directions.

2.14.4 Supported Subsets

The Unicode Standard does not require that an application be capable of interpreting and rendering all Unicode characters so as to be conformant. Many systems will have fonts for only some scripts, but not for others; sorting and other text-processing rules may be implemented for only a limited set of languages. As a result, an implementation is able to interpret a subset of characters.

The Unicode Standard provides no formalized method for identifying an implemented subset. Furthermore, such a subset is typically different for different aspects of an implementation. For example, an application may be able to read, write, and store any Unicode character and to sort one subset according to the rules of one or more languages (and the rest arbitrarily), but have access to fonts for only a single script. The same implementation may be able to render additional scripts as soon as additional fonts are installed in its environment. Therefore, the subset of interpretable characters is typically not a static concept.

Chapter 3

Conformance

This chapter defines conformance to the Unicode Standard in terms of the principles and encoding architecture it embodies. The first section defines the format for referencing the Unicode Standard and Unicode properties. The second section consists of the conformance clauses, followed by sections that define more precisely the technical terms used in those clauses. The remaining sections contain the formal algorithms that are part of conformance and referenced by the conformance clause. Additional definitions and algorithms that are part of this standard can be found in the Unicode Standard Annexes listed at the end of *Section 3.2, Conformance Requirements*.

In this chapter, conformance clauses are identified with the letter *C*. Definitions are identified with the letter *D*. Bulleted items are explanatory comments regarding definitions or subclauses.

For information on implementing best practices, see *Chapter 5, Implementation Guidelines*.

3.1 Versions of the Unicode Standard

For most character encodings, the character repertoire is fixed (and often small). Once the repertoire is decided upon, it is never changed. Addition of a new abstract character to a given repertoire creates a new repertoire, which will be treated either as an update of the existing character encoding or as a completely new character encoding.

For the Unicode Standard, by contrast, the repertoire is inherently open. Because Unicode is a universal encoding, any abstract character that could ever be encoded is a potential candidate to be encoded, regardless of whether the character is currently known.

Each new version of the Unicode Standard supersedes the previous one, but implementations—and, more significantly, data—are not updated instantly. In general, major and minor version changes include new characters, which do not create particular problems with old data. The Unicode Technical Committee will neither remove nor move characters. Characters may be *deprecated*, but this does not remove them from the standard or from existing data. The code point for a deprecated character will never be reassigned to a different character, but the use of a deprecated character is strongly discouraged. These rules make the encoded characters of a new version backward-compatible with previous versions.

Implementations should be prepared to be forward-compatible with respect to Unicode versions. That is, they should accept text that may be expressed in future versions of this standard, recognizing that new characters may be assigned in those versions. Thus they should handle incoming unassigned code points as they do unsupported characters. (See *Section 5.3, Unknown and Missing Characters*.)

A version change may also involve changes to the properties of existing characters. When this situation occurs, modifications are made to the Unicode Character Database and a new version is issued for the standard. Changes to the data files may alter program behavior that depends on them. However, such changes to properties and to data files are never made lightly. They are made only after careful deliberation by the Unicode Technical Committee has determined that there is an error, inconsistency, or other serious problem in the property assignments.

3.1.1 Stability

Each version of the Unicode Standard, once published, is absolutely stable and will *never* change. Implementations or specifications that refer to a specific version of the Unicode Standard can rely upon this stability. When implementations or specifications are upgraded to a future version of the Unicode Standard, then changes to them

may be necessary. Note that even errata and corrigenda do not formally change the text of a published version; see "Errata and Corrigenda" later in this section.

Some features of the Unicode Standard are guaranteed to be stable *across* versions. These include the names and code positions of characters, their decompositions, and several other character properties for which stability is important to implementations. See also "Stability of Properties" in *Section 3.5, Properties*. The formal statement of such stability guarantees is contained in the policies on character encoding stability found on the Unicode website. See the subsection "Policies" in *Appendix B.3, Other Unicode Online Resources*. See the discussion of backward compatibility in Section 2.5 of Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax," and the subsection "Interacting with Downlevel Systems" in *Section 5.3, Unknown and Missing Characters*.

3.1.2 Version Numbering

Version numbers for the Unicode Standard consist of three fields, denoting the major version, the minor version, and the update version, respectively. For example, "Unicode 5.2.0" indicates major version 5 of the Unicode Standard, minor version 2 of Unicode 5, and update version 0 of minor version Unicode 5.2.

To simplify implementations of Unicode version numbering, the version fields are limited to values which can be stored in a single byte. The major version is a positive integer constrained to the range 1..255. The minor and update versions are non-negative integers constrained to the range 0..255.

Additional information on the current and past versions of the Unicode Standard can be found on the Unicode website. See the subsection "Versions" in *Appendix B.3, Other Unicode Online Resources*. The online document contains the precise list of contributing files from the Unicode Character Database and the Unicode Standard Annexes, which are formally part of each version of the Unicode Standard.

Major and Minor Versions. Major and minor versions have significant additions to the standard, including, but not limited to, additions to the repertoire of encoded characters. Both are published as an updated core specification, together with associated updates to the code charts, the Unicode Standard Annexes and the Unicode Character Database. Such versions consolidate all errata and corrigenda and supersede any prior documentation for major, minor, or update versions.

A major version typically is of more importance to implementations; however, even update versions may be important to particular companies or other organizations. Major and minor versions are often synchronization points with related standards, such as with ISO/IEC 10646.

Prior to Version 5.2, minor versions of the standard were published as online amendments expressed as textual changes to the previous version, rather than as fully consolidated new editions of the core specification.

Update Version. An update version represents relatively small changes to the standard, typically updates to the data files of the Unicode Character Database. An update version never involves any additions to the character repertoire. These versions are published as modifications to the data files, and, on occasion, include documentation of small updates for selected errata or corrigenda.

Formally, each new version of the Unicode Standard supersedes all earlier versions. However, update versions generally do not obsolete the documentation of the immediately prior version of the standard.

Scheduling of Versions. Prior to Version 7.0.0, major, minor, and update versions of the Unicode Standard were published whenever the work on each new set of repertoire, properties, and documentation was finished. The emphasis was on ensuring synchronization of the major releases with corresponding major publication milestones for ISO/IEC 10646, but that practice resulted in an irregular publication schedule.

The Unicode Technical Committee changed its process as of Version 7.0.0 of the Unicode Standard, to make the publication time predictable. Major releases of the standard are now scheduled for annual publication. Further minor and update releases are not anticipated, but might occur under exceptional circumstances. This predictable, regular publication makes planning for new releases easier for most users of the standard. The detailed statements of synchronization between versions of the Unicode Standard and ISO/IEC 10646 have become somewhat more complex as a result, but in practice this has not been a problem for implementers.

3.1.3 Errata and Corrigenda

From time to time it may be necessary to publish errata or corrigenda to the Unicode Standard. Such errata and corrigenda will be published on the Unicode website. See *Appendix B.3, Other Unicode Online Resources*, for information on how to report errors in the standard.

Errata. Errata correct errors in the text or other informative material, such as the representative glyphs in the code charts. See the subsection "Updates and Errata" in *Appendix B.3, Other Unicode Online Resources*. Whenever a new major or minor version of the standard is published, all errata up to that point are incorporated into the core specification, code charts, or other components of the standard.

Corrigenda. Occasionally errors may be important enough that a corrigendum is issued prior to the next version of the Unicode Standard. Such a corrigendum does

not change the contents of the previous version. Instead, it provides a mechanism for an implementation, protocol, or other standard to cite the previous version of the Unicode Standard with the corrigendum applied. If a citation does not specifically mention the corrigendum, the corrigendum does not apply. For more information on citing corrigenda, see "Versions" in *Appendix B.3, Other Unicode Online Resources*.

3.1.4 References to the Unicode Standard

The documents associated with the major, minor, and update versions are called the major reference, minor reference, and update reference, respectively. For example, consider Unicode Version 3.1.1. The major reference for that version is *The Unicode Standard, Version 3.0* (ISBN 0-201-61633-5). The minor reference is Unicode Standard Annex #27, "The Unicode Standard, Version 3.1." The update reference is Unicode Version 3.1.1. The exact list of contributory files, Unicode Standard Annexes, and Unicode Character Database files can be found at Enumerated Version 3.1.1.

The reference for *this* version, Version 16.0.0, of the Unicode Standard, is

> The Unicode Consortium. The Unicode Standard, Version 16.0.0, defined by: *The Unicode Standard, Version 16.0* (South San Francisco: The Unicode Consortium, 2024. ISBN 978-1-936213-34-4)

References to an update (or minor version prior to Version 5.2.0) include a reference to both the major version and the documents modifying it. For the standard citation format for other versions of the Unicode Standard, see "Versions" in *Appendix B.3, Other Unicode Online Resources*.

3.1.5 Precision in Version Citation

Because Unicode has an open repertoire with relatively frequent updates, it is important not to over-specify the version number. Wherever the precise behavior of all Unicode characters needs to be cited, the full three-field version number should be used, as in the first example below. However, trailing zeros are often omitted, as in the second example. In such a case, writing 3.1 is in all respects equivalent to writing 3.1.0.

1. The Unicode Standard, Version 3.1.1

2. The Unicode Standard, Version 3.1

3. The Unicode Standard, Version 3.0 or later

4. The Unicode Standard

Where some basic level of content is all that is important, phrasing such as in the third example can be used. Where the important information is simply the overall

architecture and semantics of the Unicode Standard, the version can be omitted entirely, as in example 4.

3.1.6 References to Unicode Character Properties

Properties and property values have defined names and abbreviations, such as

Property: General_Category (gc)

Property Value: Uppercase_Letter (Lu)

To reference a given property and property value, these aliases are used, as in this example:

The property value Uppercase_Letter from the General_Category property, as specified in Version 16.0.0 of the Unicode Standard.

Then cite that version of the standard, using the standard citation format that is provided for each version of the Unicode Standard.

When referencing multi-word properties or property values, it is permissible to omit the underscores in these aliases or to replace them by spaces.

When referencing a Unicode character property, it is customary to prepend the word "Unicode" to the name of the property, unless it is clear from context that the Unicode Standard is the source of the specification.

3.1.7 References to Unicode Algorithms

A reference to a Unicode algorithm must specify the name of the algorithm or its abbreviation, followed by the version of the Unicode Standard, as in this example:

The Unicode Bidirectional Algorithm, as specified in Version 16.0.0 of the Unicode Standard.

See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm,"

https://www.unicode.org/reports/tr9/tr9-50.html

Where algorithms allow tailoring, the reference must state whether any such tailorings were applied or are applicable. For algorithms contained in a Unicode Standard Annex, the document itself and its location on the Unicode website may be cited as the location of the specification.

When referencing a Unicode algorithm it is customary to prepend the word "Unicode" to the name of the algorithm, unless it is clear from the context that the Unicode Standard is the source of the specification.

Omitting a version number when referencing a Unicode algorithm may be appropriate when such a reference is meant as a generic reference to the overall algorithm. Such a generic reference may also be employed in the sense of latest available version of the algorithm. However, for specific and detailed conformance claims for Unicode algorithms, generic references are generally not sufficient, and a full version number must accompany the reference.

3.2 Conformance Requirements

This section presents the clauses specifying the formal conformance requirements for processes implementing this version of the Unicode Standard.

In addition to this core specification, the Unicode Standard, Version 16.0.0, includes a number of Unicode Standard Annexes (UAXes) and the Unicode Character Database. At the end of this section there is a list of those annexes that are considered an integral part of the Unicode Standard, Version 16.0.0, and therefore covered by these conformance requirements.

The Unicode Character Database contains an extensive specification of normative and informative character properties completing the formal definition of the Unicode Standard. See *Chapter 4, Character Properties*, for more information.

Not all conformance requirements are relevant to all implementations at all times because implementations may not support the particular characters or operations for which a given conformance requirement may be relevant. See *Section 2.14, Conforming to the Unicode Standard*, for more information.

In this section, conformance clauses are identified with the letter *C*.

3.2.1 Code Points Unassigned to Abstract Characters

C1 *A process shall not interpret a high-surrogate code point or a low-surrogate code point as an abstract character.*

- The high-surrogate and low-surrogate code points are designated for surrogate code units in the UTF-16 character encoding form. They are unassigned to any abstract character.

C2 *A process shall not interpret a noncharacter code point as an abstract character.*

- The noncharacter code points may be used internally, such as for sentinel values or delimiters, but should not be exchanged publicly.

C3 *A process shall not interpret an unassigned code point as an abstract character.*

- This clause does not preclude the assignment of certain generic semantics to unassigned code points (for example, rendering with a glyph to indicate the position within a character block) that allow for graceful behavior in the presence of code points that are outside a supported subset.

- Unassigned code points may have default property values. (See D26.)

• Code points whose use has not yet been designated may be assigned to abstract characters in future versions of the standard. Because of this fact, due care in the handling of generic semantics for such code points is likely to provide better robustness for implementations that may encounter data based on future versions of the standard.

3.2.2 Interpretation

Interpretation of characters is the key conformance requirement for the Unicode Standard, as it is for any coded character set standard. In legacy character set standards, the single conformance requirement is generally stated in terms of the interpretation of bit patterns used as characters. Conforming to a particular standard requires interpreting bit patterns used as characters according to the list of character names and the glyphs shown in the associated code table that form the bulk of that standard.

Interpretation of characters is a more complex issue for the Unicode Standard. It includes the core issue of interpreting code points used as characters according to the names and representative glyphs shown in the code charts, of course. However, the Unicode Standard also specifies character properties, behavior, and interactions between characters. Such information about characters is considered an integral part of the "character semantics established by this standard."

Information about the properties, behavior, and interactions between Unicode characters is provided in the Unicode Character Database and in the Unicode Standard Annexes. Additional information can be found throughout the other chapters of this core specification for the Unicode Standard. However, because of the need to keep extended discussions of scripts, sets of symbols, and other characters readable, material in other chapters is not always labeled as to its normative or informative status. In general, supplementary semantic information about a character is considered normative when it contributes directly to the identification of the character or its behavior. Additional information provided about the history of scripts, the languages which use particular characters, and so forth, is merely informative. Thus, for example, the rules about Devanagari rendering specified in *Section 12.1, Devanagari*, or the rules about Arabic character shaping specified in *Section 9.2, Arabic*, are normative: they spell out important details about how those characters behave in conjunction with each other that is necessary for proper and complete interpretation of the respective Unicode characters covered in each section.

> *C4   A process shall interpret a coded character sequence according to the character semantics established by this standard, if that process does interpret that coded character sequence.*

• This restriction does not preclude internal transformations that are never visible external to the process.

C5   *A process shall not assume that it is required to interpret any particular coded character sequence.*

- Processes that interpret only a subset of Unicode characters are allowed; there is no blanket requirement to interpret *all* Unicode characters.

- Any means for specifying a subset of characters that a process can interpret is outside the scope of this standard.

- The semantics of a private-use code point is outside the scope of this standard.

- Although these clauses are not intended to preclude enumerations or specifications of the characters that a process or system is able to interpret, they do separate supported subset enumerations from the question of conformance. In actuality, any system may occasionally receive an unfamiliar character code that it is unable to interpret.

C6   *A process shall not assume that the interpretations of two canonical-equivalent character sequences are distinct.*

- The implications of this conformance clause are twofold. First, a process is never required to give different interpretations to two different, but canonical-equivalent character sequences. Second, no process can assume that another process will make a distinction between two different, but canonical-equivalent character sequences.

- Ideally, an implementation would always interpret two canonical-equivalent character sequences identically. There are practical circumstances under which implementations may reasonably distinguish them.

- Even processes that normally do not distinguish between canonical-equivalent character sequences can have reasonable exception behavior. Some examples of this behavior include graceful fallback processing by processes unable to support correct positioning of nonspacing marks; "Show Hidden Text" modes that reveal memory representation structure; and the choice of ignoring collating behavior of combining character sequences that are not part of the repertoire of a specified language (see <u>*Section 5.12, Strategies for Handling Nonspacing Marks*</u>).

3.2.3 Modification

C7   *When a process purports not to modify the interpretation of a valid coded character sequence, it shall make no change to that coded character sequence other than the possible replacement of character sequences by their canonical-equivalent sequences.*

- Replacement of a character sequence by a compatibility-equivalent sequence *does* modify the interpretation of the text.

- Replacement or deletion of a character sequence that the process cannot or does not interpret *does* modify the interpretation of the text.

- Changing the bit or byte ordering of a character sequence when transforming it between different machine architectures does not modify the interpretation of the text.

- Changing a valid coded character sequence from one Unicode character encoding form to another does not modify the interpretation of the text.

- Changing the byte serialization of a code unit sequence from one Unicode character encoding scheme to another does not modify the interpretation of the text.

- If a noncharacter that does not have a specific internal use is unexpectedly encountered in processing, an implementation may signal an error or replace the noncharacter with U+FFFD REPLACEMENT CHARACTER. If the implementation chooses to replace, delete or ignore a noncharacter, such an action constitutes a modification in the interpretation of the text. In general, a noncharacter should be treated as an unassigned code point. For example, an API that returned a character property value for a noncharacter would return the same value as the default value for an unassigned code point.

- Note that security problems can result if noncharacter code points are removed from text received from external sources. For more information, see *Section 23.7, Noncharacters*, and Unicode Technical Report #36, "Unicode Security Considerations."

- All processes and higher-level protocols are required to abide by conformance clause C7 at a minimum. However, higher-level protocols may define additional equivalences that do not constitute modifications under that protocol. For example, a higher-level protocol may allow a sequence of spaces to be replaced by a single space.

- There are important security issues associated with the correct interpretation and display of text. For more information, see Unicode Technical Report #36, "Unicode Security Considerations."

3.2.4 Character Encoding Forms

C8 *When a process interprets a code unit sequence which purports to be in a Unicode character encoding form, it shall interpret that code unit sequence according to the corresponding code point sequence.*

- The specification of the code unit sequences for UTF-8 is given in D92.

- The specification of the code unit sequences for UTF-16 is given in D91.

- The specification of the code unit sequences for UTF-32 is given in D90.

C9   *When a process generates a code unit sequence which purports to be in a Unicode character encoding form, it shall not emit ill-formed code unit sequences.*

- The definition of each Unicode character encoding form specifies the ill-formed code unit sequences in the character encoding form. For example, the definition of UTF-8 (D92) specifies that code unit sequences such as <C0 AF> are ill-formed.

C10   *When a process interprets a code unit sequence which purports to be in a Unicode character encoding form, it shall treat ill-formed code unit sequences as an error condition and shall not interpret such sequences as characters.*

- For example, in UTF-8 every code unit of the form $110xxxx_2$ *must* be followed by a code unit of the form $10xxxxxx_2$. A sequence such as $110xxxxx_2$ $0xxxxxxx_2$ is ill-formed and must never be generated. When faced with this ill-formed code unit sequence while transforming or interpreting text, a conformant process must treat the first code unit $110xxxxx_2$ as an illegally terminated code unit sequence —for example, by signaling an error or representing the code unit with a marker such as U+FFFD REPLACEMENT CHARACTER.

- Silently ignoring ill-formed sequences is strongly discouraged because joining text from before and after the ill-formed sequence can cause the resulting text to take a new meaning. This result would be especially dangerous in the context of textual formats that carry embedded program code, such as JavaScript.

- Conformant processes cannot interpret ill-formed code unit sequences. However, the conformance clauses do not prevent processes from operating on code unit sequences that do not purport to be in a Unicode character encoding form. For example, for performance reasons a low-level string operation may simply operate directly on code units, without interpreting them as characters. See, especially, the discussion under D89.

- Utility programs are not prevented from operating on "mangled" text. For example, a UTF-8 file could have had CRLF sequences introduced at every 80 bytes by a bad mailer program. This could result in some UTF-8 byte sequences being interrupted by CRLFs, producing illegal byte sequences. This mangled text is no longer UTF-8. It is permissible for a conformant program to repair such text, recognizing that the mangled text was originally well-formed UTF-8 byte sequences. However, such repair of mangled data is a special case, and it must not be used in circumstances where it would cause security problems. There are important security issues associated with encoding conversion, especially with the conversion of malformed text. For more information, see Unicode Technical Report #36, "Unicode Security Considerations."

3.2.5 Character Encoding Schemes

C11 *When a process interprets a byte sequence which purports to be in a Unicode character encoding scheme, it shall interpret that byte sequence according to the byte order and specifications for the use of the byte order mark established by this standard for that character encoding scheme.*

- Machine architectures differ in *ordering* in terms of whether the most significant byte or the least significant byte comes first. These sequences are known as "big-endian" and "little-endian" orders, respectively.

- For example, when using UTF-16LE, pairs of bytes are interpreted as UTF-16 code units using the little-endian byte order convention, and any initial <FF FE> sequence is interpreted as U+FEFF ZERO WIDTH NO-BREAK SPACE (part of the text), rather than as a byte order mark (not part of the text). (See D97.)

3.2.6 Bidirectional Text

C12 *A process that displays text containing supported right-to-left characters or embedding codes shall display all visible representations of characters (excluding format characters) in the same order as if the Bidirectional Algorithm had been applied to the text, unless tailored by a higher-level protocol as permitted by the specification.*

- The Bidirectional Algorithm is specified in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

3.2.7 Normalization Forms

C13 *A process that produces Unicode text that purports to be in a Normalization Form shall do so in accordance with the specifications in Section 3.11, Normalization Forms.*

C14 *A process that tests Unicode text to determine whether it is in a Normalization Form shall do so in accordance with the specifications in Section 3.11, Normalization Forms.*

C15 *A process that purports to transform text into a Normalization Form must be able to produce the results of the conformance test specified in Unicode Standard Annex #15, "Unicode Normalization Forms."*

- This means that when a process uses the input specified in the conformance test, its output must match the expected output of the test.

3.2.8 Normative References

C16 *Normative references to the Unicode Standard itself, to property aliases, to property value aliases, or to Unicode algorithms shall follow the formats specified in Section 3.1, Versions of the Unicode Standard.*

C17 *Higher-level protocols shall not make normative references to provisional properties.*

• Higher-level protocols may make normative references to informative properties.

3.2.9 Unicode Algorithms

C18 *If a process purports to implement a Unicode algorithm, it shall conform to the specification of that algorithm in the standard, including any tailoring by a higher-level protocol as permitted by the specification.*

• The term *Unicode algorithm* is defined at D17.

• An implementation claiming conformance to a Unicode algorithm need only guarantee that it produces the same results as those specified in the logical description of the process; it is not required to follow the actual described procedure in detail. This allows room for alternative strategies and optimizations in implementation.

C19 *The specification of an algorithm may prohibit or limit tailoring by a higher-level protocol. If a process that purports to implement a Unicode algorithm applies a tailoring, that fact must be disclosed.*

• For example, the algorithms for normalization and canonical ordering are not tailorable. The Bidirectional Algorithm allows some tailoring by higher-level protocols. The Unicode Default Case algorithms may be tailored without limitation.

3.2.10 Default Casing Algorithms

C20 *An implementation that purports to support Default Case Conversion, Default Case Detection, or Default Caseless Matching shall do so in accordance with the definitions and specifications in Section 3.13, Default Case Algorithms.*

• A conformant implementation may perform casing operations that are different from the default algorithms, perhaps tailored to a particular orthography, so long as the fact that a tailoring is applied is disclosed.

3.2.11 Unicode Standard Annexes

The following standard annexes are approved and considered part of Version 16.0 of the Unicode Standard. These annexes may contain either normative or informative material, or both. Any reference to Version 16.0 of the standard automatically includes Version 16.0 of these standard annexes.

- UAX #9: Unicode Bidirectional Algorithm
- UAX #11: East Asian Width
- UAX #14: Unicode Line Breaking Algorithm
- UAX #15: Unicode Normalization Forms
- UAX #24: Unicode Script Property
- UAX #29: Unicode Text Segmentation
- UAX #31: Unicode Identifier and Pattern Syntax
- UAX #34: Unicode Named Character Sequences
- UAX #38: Unicode Han Database (Unihan)
- UAX #41: Common References for Unicode Standard Annexes
- UAX #42: Unicode Character Database in XML
- UAX #44: Unicode Character Database
- UAX #45: U-Source Ideographs
- UAX #50: Unicode Vertical Text Layout
- UAX #53: Unicode Arabic Mark Rendering
- UAX #57: Unicode Egyptian Hieroglyph Database (Unikemet)

Conformance to the Unicode Standard requires conformance to the specifications contained in these annexes, as detailed in the conformance clauses listed earlier in this section.

3.3 Semantics

3.3.1 Definitions

This and the following sections more precisely define the terms that are used in the conformance clauses.

3.3.2 Character Identity and Semantics

D1   *Normative behavior:* The normative behaviors of the Unicode Standard consist of the following list or any other behaviors specified in the conformance clauses:

- Character combination

- Canonical decomposition

- Compatibility decomposition

- Canonical ordering behavior

- Bidirectional behavior, as specified in the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm")

- Conjoining jamo behavior, as specified in *Section 3.12, Conjoining Jamo Behavior*

- Variation selection, as specified in *Section 23.4, Variation Selectors*

- Normalization, as specified in *Section 3.11, Normalization Forms*

- Default casing, as specified in *Section 3.13, Default Case Algorithms*

D2   *Character identity:* The identity of a character is established by its character name and representative glyph in the code charts.

- A character may have a broader range of use than the most literal interpretation of its name might indicate; the coded representation, name, and representative glyph need to be assessed in context when establishing the identity of a character. For example, U+002E FULL STOP can represent a sentence period, an abbreviation period, a decimal number separator in English, a thousands number separator in German, and so on. The character name itself is unique, but may be misleading. See "Character Names" in *Section 24.1, Character Names List*.

- Consistency with the representative glyph does not require that the images be identical or even graphically similar; rather, it means that both images are generally recognized to be representations of the same character. Representing the character U+0061 LATIN SMALL LETTER A by the glyph "X" would violate its character identity.

D3 *Character semantics:* The semantics of a character are determined by its iden-
tity, normative properties, and behavior.

- Some normative behavior is default behavior; this behavior can be overridden by
higher-level protocols. However, in the absence of such protocols, the behavior
must be observed so as to follow the character semantics.

- The character combination properties and the canonical ordering behavior
cannot be overridden by higher-level protocols. The purpose of this constraint is
to guarantee that the order of combining marks in text and the results of
normalization are predictable.

D4 *Character name:* A unique string used to identify each abstract character
encoded in the standard.

- The character names in the Unicode Standard match those of the English
edition of ISO/IEC 10646.

- Character names are immutable and cannot be overridden; they are stable
identifiers. For more information, see *Section 4.8, Name*.

- The name of a Unicode character is also formally a character property in the
Unicode Character Database. Its long property alias is "Name" and its short
property alias is "na". Its value is the unique string label associated with the
encoded character.

- The detailed specification of the Unicode character names, including rules for
derivation of some ranges of characters, is given in *Section 4.8, Name*. That
section also describes the relationship between the normative value of the Name
property and the contents of the corresponding data field in UnicodeData.txt in
the Unicode Character Database.

D5 *Character name alias:* An additional unique string identifier, other than the
character name, associated with an encoded character in the standard.

- Character name aliases are assigned when there is a serious clerical defect with
a character name, such that the character name itself may be misleading
regarding the identity of the character. A character name alias constitutes an
alternate identifier for the character.

- Character name aliases are also assigned to provide string identifiers for control
codes and to recognize widely used alternative names and abbreviations for
control codes, format characters and other special-use characters.

- Character name aliases are unique within the common namespace shared by
character names, character name aliases, named character sequences, and code
point labels.

- More than one character name alias may be assigned to a given Unicode character. For example, the control code U+000D is given a character name alias for its ISO 6429 control function as CARRIAGE RETURN, but is also given a character name alias for its widely used abbreviation "CR".

- Character name aliases are a formal, normative part of the standard and should be distinguished from the informative, editorial aliases provided in the code charts. See *Section 24.1, Character Names List*, for the notational conventions used to distinguish the two.

D6 *Namespace:* A set of names together with name matching rules, so that all names are distinct under the matching rules.

- Within a given namespace all names must be unique, although the same name may be used with a different meaning in a different namespace.

- Character names, character name aliases, named character sequences, and code point labels share a single namespace in the Unicode Standard.

3.4 Characters and Encoding

 D7 *Abstract character:* A unit of information used for the organization, control, or representation of textual data.

 • When representing data, the nature of that data is generally symbolic as opposed to some other kind of data (for example, aural or visual). Examples of such symbolic data include letters, ideographs, digits, punctuation, technical symbols, and dingbats.

 • An abstract character has no concrete form and should not be confused with a *glyph*.

 • An abstract character does not necessarily correspond to what a user thinks of as a "character" and should not be confused with a *grapheme*.

 • The abstract characters encoded by the Unicode Standard are known as Unicode abstract characters.

 • Abstract characters not directly encoded by the Unicode Standard can often be represented by the use of combining character sequences.

 D8 *Abstract character sequence:* An ordered sequence of one or more abstract characters.

 D9 *Unicode codespace:* A range of integers from 0 to 10FFFF_{16}.

 • This particular range is defined for the codespace in the Unicode Standard. Other character encoding standards may use other codespaces.

 D10 *Code point:* Any value in the Unicode codespace.

 • A code point is also known as a *code position*.

 • See D77 for the definition of *code unit*.

 D10a *Code point type:* Any of the seven fundamental classes of code points in the standard: Graphic, Format, Control, Private-Use, Surrogate, Noncharacter, Reserved.

 • See *Table 2-3* for a summary of the meaning and use of each class.

 • For Noncharacter, see also D14 Noncharacter.

 • For Reserved, see also D15 Reserved code point.

 • For Private-Use, see also D49 Private-use code point.

 • For Surrogate, see also D71 High-surrogate code point and D73 Low-surrogate code point.

D10b *Block:* A named range of code points used to organize the allocation of characters.

- The exact list of blocks defined for each version of the Unicode Standard is specified by the data file Blocks.txt in the Unicode Character Database.

- The range for each defined block is specified by Field 0 in Blocks.txt; for example, "0000..007F".

- The ranges for blocks are non-overlapping. In other words, no code point can be contained in the range for one block and also in the range for a second distinct block.

- The range for each block is defined as a contiguous sequence. In other words, a block cannot consist of two (or more) discontiguous sequences of code points.

- Each range for a defined block starts with a value for which code point MOD 16 = 0 and terminates with a larger value for which code point MOD 16 = 15. This specification results in block ranges which always include full code point columns for code chart display. A block never starts or terminates in mid-column.

- All assigned characters are contained within ranges for defined blocks.

- Blocks may contain reserved code points, but no block contains *only* reserved code points. The majority of reserved code points are outside the ranges of defined blocks.

- A few designated code points are not contained within the ranges for defined blocks. This applies to the noncharacter code points at the last two code points of supplementary planes 1 through 14.

- The name for each defined block is specified by Field 1 in Blocks.txt; for example, "Basic Latin".

- The names for defined blocks constitute a unique namespace.

- The uniqueness rule for the block namespace is LM3, as defined in Unicode Standard Annex #44, "Unicode Character Database." In other words, casing, whitespace, hyphens, and underscores are ignored when matching strings for block names. The string "BASIC LATIN" or "Basic_Latin" would be considered as matching the name for the block named "Basic Latin".

- There is also a normative Block *property*. See *Table 3-2*. The Block property is a catalog property whose value is a string that identifies a block.

- Property value aliases for the Block property are defined in PropertyValueAliases.txt in the Unicode Character Database. The long alias defined for the Block property is always a loose match for the name of the block defined in Blocks.txt. Additional short aliases and other aliases are provided for convenience of use in regular expression syntax.

- The default value for the Block property is "No_Block". This default applies to any code point which is not contained in the range of a defined block.

For a general discussion of blocks and their relation to allocation in the Unicode Standard, see "Allocation Areas and Blocks" in *Section 2.8, Unicode Allocation*. For a general discussion of the use of blocks in the presentation of the Unicode code charts, see *Chapter 24, About the Code Charts*.

D11 *Encoded character:* An association (or mapping) between an abstract character and a code point.

- An encoded character is also referred to as a *coded character*.

- While an encoded character is formally defined in terms of the mapping between an abstract character and a code point, informally it can be thought of as an abstract character taken together with its assigned code point.

- Occasionally, for compatibility with other standards, a single abstract character may correspond to more than one code point—for example, "Å" corresponds both to U+00C5 Å LATIN CAPITAL LETTER A WITH RING ABOVE and to U+212B Å ANGSTROM SIGN.

- A single abstract character may also be *represented* by a sequence of code points —for example, *latin capital letter g with acute* may be represented by the sequence <U+0047 LATIN CAPITAL LETTER G, U+0301 COMBINING ACUTE ACCENT>, rather than being mapped to a single code point.

D12 *Coded character sequence:* An ordered sequence of one or more code points.

- A coded character sequenceis also known as a *coded character representation*.

- Normally a coded character sequence consists of a sequence of encoded characters, but it may also include noncharacters or reserved code points.

- Internally, a process may choose to make use of noncharacter code points in its coded character sequences. However, such noncharacter code points may not be interpreted as abstract characters (see conformance clause C2). Their removal by a conformant process constitutes modification of interpretation of the coded character sequence (see conformance clause C7).

- Reserved code points are included in coded character sequences, so that the conformance requirements regarding interpretation and modification are properly defined when a Unicode-conformant implementation encounters coded character sequences produced under a future version of the standard.

Unless specified otherwise for clarity, in the text of the Unicode Standard the term *character* alonedesignates an encoded character. Similarly, the term *character sequence* alone designates a coded character sequence.

D13 *Deprecated character:* A coded character whose use is strongly discouraged.

- Deprecated characters are retained in the standard indefinitely, but should not be used. They are retained in the standard so that previously conforming data stay conformant in future versions of the standard.

- Deprecated characters typically consist of characters with significant architectural problems, or ones which cause implementation problems. Some examples of characters deprecated on these grounds include U+E0001 LANGUAGE TAG (see *Section 23.9, Tag Characters*) and the alternate format characters (see *Section 23.3, Deprecated Format Characters*).

- Deprecated characters are explicitly indicated in the Unicode code charts. They are also given an explicit property value of Deprecated = True in the Unicode Character Database.

- Deprecated characters should not be confused with obsolete characters, which are historical. Obsolete characters do not occur in modern text, but they are not deprecated; their use is not discouraged.

D14 *Noncharacter:* A code point that is permanently reserved and that will never be assigned to an abstract character. Noncharacters consist of the values U+nFFFE and U+nFFFF (where n is from 0 to 10_{16}) and the values U+FDD0..U+FDEF.

- Noncharacters are not intended for interchange, but may be used by an implementation for internal purposes.

- Possible use cases include application-internal sentinel values.

- For more information, see *Section 23.7, Noncharacters*.

D15 *Reserved code point:* Any code point of the Unicode Standard that is reserved for future assignment. Also known as an *unassigned code point.*

- Surrogate code points and noncharacters are considered assigned code points, but not assigned characters.

- For a summary classification of reserved and other types of code points, see *Table 2-3*.

In general, a conforming process may indicate the presence of a code point whose use has not been designated (for example, by showing a missing glyph in rendering or by signaling an appropriate error in a streaming protocol), even though it is forbidden by the standard from *interpreting* that code point as an abstract character.

D16 *Higher-level protocol:* Any agreement on the interpretation of Unicode characters that extends beyond the scope of this standard.

- Such an agreement need not be formally announced in data; it may be implicit in the context.
- The specification of some Unicode algorithms may limit the scope of what a conformant higher-level protocol may do.

D17 *Unicode algorithm:* The logical description of a process used to achieve a specified result involving Unicode characters.

- This definition, as used in the Unicode Standard and other publications of the Unicode Consortium, is intentionally broad so as to allow precise logical description of required results, without constraining implementations to follow the precise steps of that logical description.

D18 *Named Unicode algorithm:* A Unicode algorithm that is specified in the Unicode Standard or in other standards published by the Unicode Consortium and that is given an explicit name for ease of reference.

- Named Unicode algorithms are cited in titlecase in the Unicode Standard.

Table 3-1 lists the named Unicode algorithms and indicates the locations of their specifications. Details regarding conformance to these algorithms and any restrictions they place on the scope of allowable tailoring by higher-level protocols can be found in the specifications. In some cases, a named Unicode algorithm is provided for information only. When externally referenced, a named Unicode algorithm may be prefixed with the qualifier "Unicode" to make the connection of the algorithm to the Unicode Standard and other Unicode specifications clear. Thus, for example, the Bidirectional Algorithm is generally referred to by its full name, "Unicode Bidirectional Algorithm." As much as is practical, the titles of Unicode Standard Annexes which define Unicode algorithms consist of the name of the Unicode algorithm they specify. In a few cases, named Unicode algorithms are also widely known by their acronyms, and those acronyms are also listed in *Table 3-1*.

Table 3-1. Named Unicode Algorithms

Name	Description
Canonical Ordering	*Section 3.11*
Canonical Composition	*Section 3.11*
Normalization	*Section 3.11*
Hangul Syllable Composition	*Section 3.12*
Hangul Syllable Decomposition	*Section 3.12*
Hangul Syllable Name Generation	*Section 3.12*

Default Case Conversion	*Section 3.13*
Default Case Detection	*Section 3.13*
Default Caseless Matching	*Section 3.13*
Unicode Bidirectional Algorithm (UBA)	UAX #9
Line Breaking Algorithm	UAX #14
Character Segmentation	UAX #29
Word Segmentation	UAX #29
Sentence Segmentation	UAX #29
Hangul Syllable Boundary Determination	UAX #29
Arabic Mark Transient Reordering Algorithm (AMTRA)	UAX #53
Standard Compression Scheme for Unicode (SCSU)	UTS #6
Unicode Collation Algorithm (UCA)	UTS #10

3.5 Properties

The Unicode Standard specifies many different types of character properties. This section provides the basic definitions related to character properties.

The actual values of Unicode character properties are specified in the Unicode Character Database. See _Section 4.1, Unicode Character Database_, for an overview of those data files. _Chapter 4, Character Properties_, contains more detailed descriptions of some particular, important character properties. Additional properties that are specific to particular characters (such as the definition and use of the _right-to-left override_ character or _zero width space_) are discussed in the relevant sections of this standard.

The interpretation of some properties (such as the case of a character) is independent of context, whereas the interpretation of other properties (such as directionality) is applicable to a character sequence as a whole, rather than to the individual characters that compose the sequence.

3.5.1 Types of Properties

D19 _Property:_ A named attribute of an entity in the Unicode Standard, associated with a defined set of values.

- The lists of code point and encoded character properties for the Unicode Standard are documented in Unicode Standard Annex #44, "Unicode Character Database," and in Unicode Standard Annex #38, "Unicode Han Database (Unihan)."
- The file PropertyAliases.txt in the Unicode Character Database provides a machine-readable list of the non-Unihan properties and their names.

D20 _Code point property:_ A property of code points.

- Code point properties refer to attributes of code points per se, based on architectural considerations of this standard, irrespective of any particular encoded character.
- Thus the Surrogate property and the Noncharacter property are code point properties.

D21 _Abstract character property:_ A property of abstract characters.

- Abstract character properties refer to attributes of abstract characters per se, based on their independent existence as elements of writing systems or other notational systems, irrespective of their encoding in the Unicode Standard.

- Thus the Alphabetic property, the Punctuation property, the Hex_Digit property, the Numeric_Value property, and so on are properties of abstract characters and are associated with those characters whether encoded in the Unicode Standard or in any other character encoding—or even prior to their being encoded in any character encoding standard.

D22 *Encoded character property:* A property of encoded characters in the Unicode Standard.

- For each encoded character property there is a mapping from every code point to some value in the set of values associated with that property.

Encoded character properties are defined this way to facilitate the implementation of character property APIs based on the Unicode Character Database. Typically, an API will take a property and a code point as input, and will return a value for that property as output, interpreting it as the "character property" for the "character" encoded at that code point. However, to be useful, such APIs must return meaningful values for unassigned code points, as well as for encoded characters.

In some instances an encoded character property in the Unicode Standard is exactly equivalent to a code point property. For example, the Pattern_Syntax property simply defines a range of code points that are reserved for pattern syntax. (See Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax.")

In other instances, an encoded character property directly reflects an abstract character property, but extends the domain of the property to include all code points, including unassigned code points. For Boolean properties, such as the Hex_Digit property, typically an encoded character property will be true for the encoded characters with that abstract character property and will be false for all other code points, including unassigned code points, noncharacters, private-use characters, and encoded characters for which the abstract character property is inapplicable or irrelevant.

However, in many instances, an encoded character property is semantically complex and may telescope together values associated with a number of abstract character properties and/or code point properties. The General_Category property is an example—it contains values associated with several abstract character properties (such as Letter, Punctuation, and Symbol) as well as code point properties (such as \p{gc=Cs} for the Surrogate code point property).

In the text of this standard the terms "Unicode character property," "character property," and "property" without qualifier generally refer to an encoded character property, unless otherwise indicated.

A list of the encoded character properties formally considered to be a part of the Unicode Standard can be found in PropertyAliases.txt in the Unicode Character Database. See also "Property Aliases" later in this section.

3.5.2 Property Values

D23 *Property value:* One of the set of values associated with an encoded character property.

- For example, the East_Asian_Width [EAW] property has the possible values "Narrow", "Neutral", "Wide", "Ambiguous", and "Unassigned".

A list of the values associated with encoded character properties in the Unicode Standard can be found in PropertyValueAliases.txt in the Unicode Character Database. See also "Property Aliases" later in this section.

D24 *Explicit property value:* A value for an encoded character property that is explicitly associated with a code point in one of the data files of the Unicode Character Database.

D25 *Implicit property value:* A value for an encoded character property that is given by a generic rule or by an "otherwise" clause in one of the data files of the Unicode Character Database.

- Implicit property values are used to avoid having to explicitly list values for more than 1 million code points (most of them unassigned) for every property.

3.5.3 Default Property Values

To work properly in implementations, unassigned code points must be given default property values as if they were characters, because various algorithms require property values to be assigned to every code point before they can function at all.

Default property values are not uniform across all unassigned code points, because certain ranges of code points need different values for particular properties to maximize compatibility with expected future assignments. This means that some encoded character properties have multiple default values. For example, the Bidi_Class property defines a range of unassigned code points as having the "R" value, another range of unassigned code points as having the "AL" value, and the otherwise case as having the "L" value. For information on the default values for each encoded character property, see its description in the Unicode Character Database.

Default property values for unassigned code points are normative. They should not be changed by implementations to other values.

Default property values are also provided for private-use characters. Because the interpretation of private-use characters is subject to private agreement between the parties which exchange them, most default property values for those characters are overridable by higher-level protocols, to match the agreed-upon semantics for the characters. There are important exceptions for a few properties and Unicode algorithms. See *Section 23.5, Private-Use Characters*.

D26 *Default property value:* The value (or in some cases small set of values) of a property associated with unassigned code points or with encoded characters for which the property is irrelevant.

- For example, for most Boolean properties, "false" is the default property value. In such cases, the default property value used for unassigned code points may be the same value that is used for many assigned characters as well.

- Some properties, particularly enumerated properties, specify a particular, unique value as their default value. For example, "XX" is the default property value for the Line_Break property.

- A default property value is typically omitted when listing property values to avoid having to repeat long lists of unassigned code points. The default value may instead be specified by explicit directives or in the description of the property.

- In the case of some properties with arbitrary string values, the default property value is an implied null value. For example, the fact that there is no Unicode character name for unassigned code points is equivalent to saying that the default property value for the Name property for an unassigned code point is a null string. This may also be indicated by an explicit directive.

- For properties that map from code points to string values, the default is typically the identity mapping as opposed to a constant value over a range of code points.

- In certain cases, the default property for a code point may be the value of another property for that code point, including its default property values. For example, the default property value for the Scripts_Extensions property for a given code point is a set containing a single element, the value of the Script property for that code point.

3.5.4 Classification of Properties by Their Values

D27 *Enumerated property:* A property with a small set of named values.

- As characters are added to the Unicode Standard, the set of values may need to be extended in the future, but enumerated properties have a relatively fixed set of possible values.

D28 *Closed enumeration:* An enumerated property for which the set of values is closed and will not be extended for future versions of the Unicode Standard.

- The General_Category and Bidi_Class properties are the only closed enumerations, except for the Boolean properties.

D29 *Boolean property:* A closed enumerated property whose set of values is limited to "true" and "false".

- The presence or absence of the property is the essential information.
- Boolean properties are also commonly referred to as *binary* properties.
- In the UCD, the "true" and "false" values for a Boolean property have multiple aliases. For convenience, they may be referred to with the abbreviations "T" and "F" or as "yes" and "no" (abbreviated "Y" and "N").

D30 *Numeric property:* A numeric property is a property whose value is a number that can take on any integer or real value.

- An example is the Numeric_Value property. There is no implied limit to the number of possible distinct values for the property, except the limitations on representing integers or real numbers in computers.

D31 *String-valued property:* A property whose value is a string.

- The Canonical_Decomposition property is a string-valued property.

D32 *Catalog property:* A property that is an enumerated property, typically unrelated to an algorithm, that may be extended in each successive version of the Unicode Standard.

- Examples are the Age, Block, and Script properties. Additional new values for the set of enumerated values for these properties may be added each time the standard is revised. A new value for Age is added for each new Unicode version, a new value for Block is added for each new block added to the standard, and a new value for Script is added for each new script added to the standard.

Most properties have a single value associated with each code point. However, some properties may instead associate a set of multiple different values with each code point. See *Section 5.7.6, Properties Whose Values Are Sets of Values*, in Unicode Standard Annex #44, "Unicode Character Database."

3.5.5 Property Status

Each Unicode character property has one of several different statuses: normative, informative, contributory, or provisional. Each of these statuses is formally defined

below, with some explanation and examples. In addition, normative properties can be subclassified, based on whether or not they can be overridden by conformant higher-level protocols.

The full list of currently defined Unicode character properties is provided in Unicode Standard Annex #44, "Unicode Character Database" and in Unicode Standard Annex #38, "Unicode Han Database (Unihan)." The tables of properties in those documents specify the status of each property explicitly. The data file PropertyAliases.txt provides a machine-readable listing of the character properties, except for those associated with the Unicode Han Database. The long alias for each property in PropertyAliases.txt also serves as the formal name of that property. In case of any discrepancy between the listing in PropertyAliases.txt and the listing in Unicode Standard Annex #44 or any other text of the Unicode Standard, the listing in PropertyAliases.txt should be taken as definitive. The tag for each Unihan-related character property documented in Unicode Standard Annex #38 serves as the formal name of that property.

> D33 *Normative property:* A Unicode character property used in the specification of the standard.

Specification that a character property is *normative* means that implementations which claim conformance to a particular version of the Unicode Standard and which make use of that particular property must follow the specifications of the standard for that property for the implementation to be conformant. For example, the Bidi_Class property is required for conformance whenever rendering text that requires bidirectional layout, such as Arabic or Hebrew.

Whenever a normative process depends on a property in a specified way, that property is designated as normative.

The fact that a given Unicode character property is normative does *not* mean that the values of the property will never change for particular characters. Corrections and extensions to the standard in the future may require minor changes to normative values, even though the Unicode Technical Committee strives to minimize such changes. See also "Stability of Properties" later in this section.

Some of the normative Unicode algorithms depend critically on particular property values for their behavior. Normalization, for example, defines an aspect of textual interoperability that many applications rely on to be absolutely stable. As a result, some of the normative properties disallow any kind of overriding by higher-level protocols. Thus the decomposition of Unicode characters is both normative and *not overridable*; no higher-level protocol may override these values, because to do so would result in non-interoperable results for the normalization of Unicode text. Other normative properties, such as case mapping, are *overridable* by higher-level protocols, because their intent is to provide a common basis for behavior. Neverthe-

less, they may require tailoring for particular local cultural conventions or particular implementations.

D34 *Overridable property:* A normative property whose values may be overridden by conformant higher-level protocols.

- For example, the Canonical_Decomposition property is not overridable. The Uppercase property can be overridden.

Some important normative character properties of the Unicode Standard are listed in *Table 3-2*, with an indication of which sections in the standard provide a general description of the properties and their use. Other normative properties are documented in the Unicode Character Database. In all cases, the Unicode Character Database provides the definitive list of character properties and the exact list of property value assignments for each version of the standard.

Table 3-2. Normative Character Properties

Property	Description
Bidi_Class (directionality)	UAX #9 and *Section 4.4*
Bidi_Mirrored	UAX #9 and *Section 4.7*
Bidi_Paired_Bracket	UAX #9
Bidi_Paired_Bracket_Type	UAX #9
Block	*Section 24.1*
Canonical_Combining_Class	*Section 3.11* and *Section 4.3*
Case-related properties	*Section 3.13*, *Section 4.2*, and UAX #44
Composition_Exclusion	*Section 3.11*
Decomposition_Mapping	*Section 3.7* and *Section 3.11*
Default_Ignorable_Code_Point	*Section 5.21*
Deprecated	*Section 3.1*
East_Asian_Width	*Section 18.4* and UAX #11
General_Category	*Section 4.5*
Hangul_Syllable_Type	*Section 3.12* and UAX #29
Joining_Type and Joining_Group	*Section 9.2*
Line_Break	*Section 23.1*, *Section 23.2*, and UAX #14
Name	*Section 4.8*
Noncharacter_Code_Point	*Section 23.7*
Numeric_Value	*Section 4.6*
White_Space	UAX #44

D35 *Informative property:* A Unicode character property whose values are provided for information only.

A conformant implementation of the Unicode Standard is free to use or change informative property values as it may require, while remaining conformant to the standard. An implementer always has the option of establishing a protocol to convey the fact that informative properties are being used in distinct ways.

Informative properties capture expert implementation experience. When an informative property is explicitly specified in the Unicode Character Database, its use is strongly recommended for implementations to encourage comparable behavior between implementations. Note that it is possible for an informative property in one version of the Unicode Standard to become a normative property in a subsequent version of the standard if its use starts to acquire conformance implications in some part of the standard.

Table 3-3 provides a partial list of the more important informative character properties. For a complete listing, see the Unicode Character Database.

Table 3-3. Informative Character Properties

Property	Description
Dash	*Section 6.2* and *Table 6-3*
Letter-related properties	*Section 4.10*
Mathematical	*Section 22.5*
Script	UAX #24
Space	*Section 6.2* and *Table 6-2*
Unicode_1_Name	*Section 4.9*

D35a *Contributory property:* A simple property defined merely to make the statement of a rule defining a derived property more compact or general.

Contributory properties typically consist of short lists of exceptional characters which are used as part of the definition of a more generic normative or informative property. In most cases, such properties are given names starting with "Other", as Other_Alphabetic or Other_Default_Ignorable_Code_Point.

Contributory properties are not themselves subject to stability guarantees, but they are sometimes specified in order to make it easier to state the definition of a derived property which itself is subject to a stability guarantee, such as the derived, normative identifier-related properties, XID_Start and XID_Continue. The complete list of contributory properties is documented in Unicode Standard Annex #44, "Unicode Character Database."

D36 *Provisional property:* A Unicode character property whose values are unapproved and tentative, and which may be incomplete or otherwise not in a usable state.

- Provisional properties may be removed from future versions of the standard, without prior notice.

Some of the information provided about characters in the Unicode Character Database constitutes provisional data. This data may capture partial or preliminary information. It may contain errors or omissions, or otherwise not be ready for systematic use; however, it is included in the data files for distribution partly to encourage review and improvement of the information. For example, a number of the tags in the Unihan Database file (Unihan.zip) provide provisional property values of various sorts about Han characters.

The data files of the Unicode Character Database may also contain various annotations and comments about characters, and those annotations and comments should be considered provisional. Implementations should not attempt to parse annotations and comments out of the data files and treat them as informative character properties per se.

Section 4.12, Characters with Unusual Properties, provides additional lists of Unicode characters with unusual behavior, including many format controls discussed in detail elsewhere in the standard. Although in many instances those characters and their behavior have normative implications, the particular subclassification provided in *Table 4-10* does not directly correspond to any formal definition of Unicode character properties. Therefore that subclassification itself should also be considered provisional and potentially subject to change.

3.5.6 Context Dependence

D37 *Context-dependent property:* A property that applies to a code point in the context of a longer code point sequence.

- For example, the lowercase mapping of a Greek sigma depends on the context of the surrounding characters.

D38 *Context-independent property:* A property that is not context dependent; it applies to a code point in isolation.

3.5.7 Stability of Properties

D39 *Stable transformation:* A transformation T on a property P is stable with respect to an algorithm A if the result of the algorithm on the transformed property $A(T(P))$ is the same as the original result $A(P)$ for all code points.

D40 *Stable property:* A property is stable with respect to a particular algorithm or process as long as possible changes in the assignment of property values are restricted in such a manner that the result of the algorithm on the property continues to be the same as the original result for all previously assigned code points.

- As new characters are assigned to previously unassigned code points, the replacement of any default values for these code points with actual property values must maintain stability.

D41 *Fixed property:* A property whose values (other than a default value), once associated with a specific code point, are fixed and will not be changed, except to correct obvious or clerical errors.

- For a fixed property, any default values can be replaced without restriction by actual property values as new characters are assigned to previously unassigned code points. Examples of fixed properties include Age and Hangul_Syllable_Type.

- Designating a property as fixed does not imply stability or immutability (see "Stability" in *Section 3.1, Versions of the Unicode Standard*). While the age of a character, for example, is established by the version of the Unicode Standard to which it was added, errors in the published listing of the property value could be corrected. For some other properties, even the correction of such errors is prohibited by explicit guarantees of property stability.

D42 *Immutable property:* A fixed property that is also subject to a stability guarantee preventing *any* change in the published listing of property values other than assignment of new values to formerly unassigned code points.

- An immutable property is trivially stable with respect to *all* algorithms.

- An example of an immutable property is the Unicode character name itself. Because character names are values of an immutable property, misspellings and incorrect names will *never* be corrected clerically. Any errata will be noted in a comment in the character names list and, where needed, an informative character name alias will be provided.

- When an encoded character property representing a code point property is immutable, none of its values can ever change. This follows from the fact that the code points themselves do not change, and the status of the property is unaffected by whether a particular abstract character is encoded at a code point later. An example of such a property is the Pattern_Syntax property; all values of that property are unchangeable for all code points, forever.

- In the more typical case of an immutable property, the values for existing encoded characters cannot change, but when a new character is encoded, the formerly unassigned code point changes from having a default value for the

property to having one of its nondefault values. Once that nondefault value is published, it can no longer be changed.

D43 *Stabilized property:* A property that is neither extended to new characters nor maintained in any other manner, but that is retained in the Unicode Character Database.

- A stabilized property is also a fixed property.

D44 *Deprecated property:* A property whose use by implementations is discouraged.

- One of the reasons a property may be deprecated is because a different combination of properties better expresses the intended semantics.

- Where sufficiently widespread legacy support exists for the deprecated property, not all implementations may be able to discontinue the use of the deprecated property. In such a case, a deprecated property may be extended to new characters so as to maintain it in a usable and consistent state.

Informative or normative properties in the standard will not be removed even when they are supplanted by other properties or are no longer useful. However, they may be stabilized and/or deprecated.

The complete list of stability policies which affect character properties, their values, and their aliases, is available online. See the subsection "Policies" in *Appendix B.3, Other Unicode Online Resources*.

3.5.8 Simple and Derived Properties

D45 *Simple property:* A Unicode character property whose values are specified directly in the Unicode Character Database (or elsewhere in the standard) and whose values cannot be derived from other simple properties.

D46 *Derived property:* A Unicode character property whose values are algorithmically derived from some combination of simple properties.

The Unicode Character Database lists a number of derived properties explicitly. Even though these values can be derived, they are provided as lists because the derivation may not be trivial and because explicit lists are easier to understand, reference, and implement. Good examples of derived properties include the ID_Start and ID_Continue properties, which can be used to specify a formal identifier syntax for Unicode characters. The details of how derived properties are computed can be found in the documentation for the Unicode Character Database.

3.5.9 Property Aliases

To enable normative references to Unicode character properties, formal aliases for properties and for property values are defined as part of the Unicode Character Database.

D47 *Property alias:* A unique identifier for a particular Unicode character property.

- The identifiers used for property aliases contain only ASCII alphanumeric characters or the underscore character.

- Short and long forms for each property alias are defined. The short forms are typically just two or three characters long to facilitate their use as attributes for tags in markup languages. For example, "General_Category" is the long form and "gc" is the short form of the property alias for the General Category property. The long form serves as the formal name for the character property.

- Property aliases are defined in the file PropertyAliases.txt that lists all of the non-Unihan properties that are part of each version of the standard. The Unihan properties are listed in Unicode Standard Annex #38, "Unicode Han Database (Unihan)."

- Property aliases of normative properties are themselves normative.

D48 *Property value alias:* A unique identifier for a particular enumerated value for a particular Unicode character property.

- The identifiers used for property value aliases contain only ASCII alphanumeric characters or the underscore character, or have the special value "n/a".

- Short and long forms for property value aliases are defined. For example, "Currency_Symbol" is the long form and "Sc" is the short form of the property value alias for the currency symbol value of the General Category property.

- Property value aliases are defined in the file PropertyValueAliases.txt in the Unicode Character Database.

- Property value aliases are unique identifiers only in the context of the particular property with which they are associated. The same identifier string might be associated with an entirely different value for a different property. The combination of a property alias and a property value alias is, however, guaranteed to be unique.

- Property value aliases referring to values of normative properties are themselves normative.

The property aliases and property value aliases can be used, for example, in XML formats of property data, for regular-expression property tests, and in other programmatic textual descriptions of Unicode property data. Thus "gc = Lu" is a formal way

of specifying that the General Category of a character (using the property alias "gc") has the value of being an uppercase letter (using the property value alias "Lu").

3.5.10 Private Use

D49 *Private-use code point:* Code points in the ranges U+E000..U+F8FF, U+F0000.. U+FFFFD, and U+100000..U+10FFFD.

- Private-use code points are considered to be assigned characters, but the abstract characters associated with them have no interpretation specified by this standard. They can be given any interpretation by conformant processes.
- Private-use code points are given default property values, but these default values are overridable by higher-level protocols that give those private-use code points a specific interpretation. See *Section 23.5, Private-Use Characters*.

3.6 Combination

3.6.1 Combining Character Sequences

D50 *Graphic character:* A character with the General Category of Letter (L), Combining Mark (M), Number (N), Punctuation (P), Symbol (S), or Space Separator (Zs).

- Graphic characters specifically exclude the line and paragraph separators (Zl, Zp), as well as the characters with the General Category of Other (Cn, Cs, Cc, Cf).

- The interpretation of private-use characters (Co) as graphic characters or not is determined by the implementation.

- For more information, see *Chapter 2, General Structure*, especially *Section 2.4, Code Points and Characters*, and *Table 2-3*.

D51 *Base character:* Any graphic character except for those with the General Category of Combining Mark (M).

- Most Unicode characters are base characters. In terms of General Category values, a base character is any code point that has one of the following categories: Letter (L), Number (N), Punctuation (P), Symbol (S), or Space Separator (Zs).

- Base characters do not include control characters or format controls.

- Base characters are independent graphic characters, but this does not preclude the presentation of base characters from adopting different contextual forms or participating in ligatures.

- The interpretation of private-use characters (Co) as base characters or not is determined by the implementation. However, the default interpretation of private-use characters should be as base characters, in the absence of other information.

D51a *Extended base:* Any base character, or any standard Korean syllable block.

- This term is defined to take into account the fact that sequences of Korean conjoining jamo characters behave as if they were a single Hangul syllable character, so that the entire sequence of jamos constitutes a base.

- For the definition of standard Korean syllable block, see D134 in *Section 3.12, Conjoining Jamo Behavior*.

D52 *Combining character:* A character with the General Category of Combining Mark (M).

- Combining characters consist of all characters with the General Category values of Spacing Combining Mark (Mc), Nonspacing Mark (Mn), and Enclosing Mark (Me).

- All characters with non-zero canonical combining class are combining characters, but the reverse is not the case: there are combining characters with a zero canonical combining class.

- The interpretation of private-use characters (Co) as combining characters or not is determined by the implementation.

- These characters are not normally used in isolation unless they are being described. They include such characters as accents, diacritics, Hebrew points, Arabic vowel signs, and Indic matras.

- The graphic positioning of a combining character depends on the last preceding base character, unless they are separated by a character that is neither a combining character nor either ZERO WIDTH JOINER or ZERO WIDTH NON-JOINER. The combining character is said to *apply* to that base character.

- There may be no such base character, such as when a combining character is at the start of text or follows a control or format character—for example, a carriage return, tab, or RIGHT-TO-LEFT MARK. In such cases, the combining characters are called *isolated combining characters*.

- With isolated combining characters or when a process is unable to perform graphical combination, a process may present a combining character without graphical combination; that is, it may present it as if it were a base character.

- The representative images of combining characters are depicted with a dotted circle in the code charts. When presented in graphical combination with a preceding base character, that base character is intended to appear in the position occupied by the dotted circle.

D53 *Nonspacing mark:* A combining character with the General Category of Nonspacing Mark (Mn) or Enclosing Mark (Me).

- The position of a nonspacing mark in presentation depends on its base character. It generally does not consume space along the visual baseline in and of itself.

- Such characters may be large enough to affect the placement of their base character relative to preceding and succeeding base characters. For example, a circumflex applied to an "i" may affect spacing ("î"), as might the character U+20DD COMBINING ENCLOSING CIRCLE.

D54 *Enclosing mark:* A nonspacing mark with the General Category of Enclosing Mark (Me).

- Enclosing marks are a subclass of nonspacing marks that surround a base character, rather than merely being placed over, under, or through it.

D55 *Spacing mark:* A combining character that is not a nonspacing mark.

- Examples include U+093F DEVANAGARI VOWEL SIGN I. In general, the behavior of spacing marks does not differ greatly from that of base characters.

- Spacing marks such as U+0BCA TAMIL VOWEL SIGN O may be rendered on both sides of a base character, but are not enclosing marks.

D56 *Combining character sequence:* A maximal character sequence consisting of either a base character followed by a sequence of one or more characters where each is a combining character, ZERO WIDTH JOINER, or ZERO WIDTH NON-JOINER; or a sequence of one or more characters where each is a combining character, ZERO WIDTH JOINER, or ZERO WIDTH NON-JOINER.

- When identifying a combining character sequence in Unicode text, the definition of the combining character sequence is applied maximally. For example, in the sequence <c, dot-below, caron, acute, a>, the entire sequence <c, dot-below, caron, acute> is identified as the combining character sequence, rather than the alternative of identifying <c, dot-below> as a combining character sequence followed by a separate (defective) combining character sequence <caron, acute>.

- Any character other than a combining mark (gc=M), ZWJ, or ZWNJ interrupts the combining character sequence. This applies even to default ignorable code points that are not also combining marks, such as U+2060 WORD JOINER or U+2064 INVISIBLE PLUS (see also *Section 5.21, Ignoring Characters in Processing*).

- A two-character sequence consisting of an initial graphic character followed by a variation selector, and satisfying additional constraints, is a variation sequence. See *Section 23.4, Variation Selectors*. Because any variation selector is a combining character, a variation sequence is either a combining character sequence, or it is a subsequence of a longer combining character sequence. For example, the sequence <0030, FE00, 20E3> represents a variant of the digit zero, followed by an enclosing keycap. A variation sequence can be a non-initial subsequence within a combining mark sequence. For example, the sequence <1000, FE00, 1031, FE00> is a single combining mark sequence with two variation sequences representing variants of the base character MYANMAR LETTER KA and the combining mark MYANMAR VOWEL SIGN E.>

D56a *Extended combining character sequence:* A maximal character sequence consisting of either an extended base followed by a sequence of one or more characters where each is a combining character, ZERO WIDTH JOINER, or ZERO WIDTH NON-JOINER; or a sequence of one or more characters where each is a combining character, ZERO WIDTH JOINER, or ZERO WIDTH NON-JOINER.

- Combining character sequence is commonly abbreviated as CCS, and extended combining character sequence is commonly abbreviated as ECCS.

D57 *Defective combining character sequence:* A combining character sequence that does not start with a base character.

- Defective combining character sequences occur when a sequence of combining characters appears at the start of a string or follows a control or format character. Such sequences are defective from the point of view of handling of combining marks, but are not *ill-formed*. (See D84.)

3.6.2 Grapheme Clusters

D58 *Grapheme base:* A character with the property Grapheme_Base, or any standard Korean syllable block.

- Characters with the property Grapheme_Base include all base characters (with the exception of U+FF9E..U+FF9F) plus most spacing marks.

- The concept of a grapheme base is introduced to simplify discussion of the graphical application of nonspacing marks to other elements of text. A grapheme base may consist of a spacing (combining) mark, which distinguishes it from a base character per se. A grapheme base may also itself consist of a sequence of characters, in the case of the standard Korean syllable block.

- For the definition of standard Korean syllable block, see D134 in *Section 3.12, Conjoining Jamo Behavior*.

D59 *Grapheme extender:* A character with the property Grapheme_Extend.

- Grapheme extender characters consist of all nonspacing marks, ZERO WIDTH JOINER, ZERO WIDTH NON-JOINER, U+FF9E HALFWIDTH KATAKANA VOICED SOUND MARK, U+FF9F HALFWIDTH KATAKANA SEMI-VOICED SOUND MARK, and a small number of spacing marks.

- A grapheme extender can be conceived of primarily as the kind of nonspacing graphical mark that is applied above or below another spacing character.

- ZERO WIDTH JOINER and ZERO WIDTH NON-JOINER are formally defined to be grapheme extenders so that their presence does not break up a sequence of other grapheme extenders.

- The small number of spacing marks that have the property Grapheme_Extend are all the second parts of a two-part combining mark.

- The set of characters with the Grapheme_Extend property and the set of characters with the Grapheme_Base property are disjoint, by definition.

- The Grapheme_Extend property is used in the derivation of the set of characters with the value Grapheme_Cluster_Break = Extend, but is not identical to it. See Section 3, "Grapheme Cluster Boundaries" in UAX #29 for details.

D60 *Grapheme cluster:* The text between grapheme cluster boundaries as specified by Unicode Standard Annex #29, "Unicode Text Segmentation."

- This definition of "grapheme cluster" is generic. The specification of grapheme cluster boundary segmentation in UAX #29 includes two alternatives, for "extended grapheme clusters" and for "legacy grapheme clusters." Furthermore, the segmentation algorithm in UAX #29 is tailorable.

- The grapheme cluster represents a horizontally segmentable unit of text, consisting of some grapheme base (which may consist of a Korean syllable) together with any number of nonspacing marks applied to it.

- A grapheme cluster is similar, but not identical to a combining character sequence. A combining character sequence starts with a base character and extends across any subsequent sequence of combining marks, *nonspacing* or *spacing*. A combining character sequence is most directly relevant to processing issues related to normalization, comparison, and searching.

- A grapheme cluster typically starts with a *grapheme base* and then extends across any subsequent sequence of *nonspacing* marks. A grapheme cluster is most directly relevant to text rendering and processes such as cursor placement and text selection in editing, but may also be relevant to comparison and searching.

- For many processes, a grapheme cluster behaves as if it were a single character with the same properties as its grapheme base. Effectively, nonspacing marks apply graphically to the base, but do not change its properties. For example, <x, macron> behaves in line breaking or bidirectional layout as if it were the character x.

D61 *Extended grapheme cluster:* The text between extended grapheme cluster boundaries as specified by Unicode Standard Annex #29, "Unicode Text Segmentation."

- Extended grapheme clusters are defined in a parallel manner to legacy grapheme clusters, but also include sequences of *spacing* marks.

- Grapheme clusters and extended grapheme clusters may not have any particular linguistic significance, but are used to break up a string of text into units for processing.

- Grapheme clusters and extended grapheme clusters may be adjusted for particular processing requirements, by tailoring the rules for grapheme cluster segmentation specified in Unicode Standard Annex #29, "Unicode Text Segmentation."

3.6.3 Application of Combining Marks

A number of principles in the Unicode Standard relate to the application of combining marks. These principles are listed in this section, with an indication of which are considered to be normative and which are considered to be guidelines.

In particular, guidelines for rendering of combining marks in conjunction with other characters should be considered as appropriate for defining default rendering behavior, in the absence of more specific information about rendering. It is often the case that combining marks in complex scripts or even particular, general-use nonspacing marks will have rendering requirements that depart significantly from the general guidelines. Rendering processes should, as appropriate, make use of available information about specific typographic practices and conventions so as to produce best rendering of text.

To help in the clarification of the principles regarding the application of combining marks, a distinction is made between *dependence* and *graphical application*.

D61a Dependence: A combining mark is said to depend on its associated base character.

- The associated base character is the base character in the combining character sequence that a combining mark is part of.

- A combining mark in a defective combining character sequence has no associated base character and thus cannot be said to depend on any particular base character. This is one of the reasons why fallback processing is required for defective combining character sequences.

- Dependence concerns *all* combining marks, including spacing marks and combining marks that have no visible display.

D61b Graphical application: A nonspacing mark is said to *apply* to its associated grapheme base.

- The associated grapheme base is the grapheme base in the grapheme cluster that a nonspacing mark is part of.

- A nonspacing mark in a defective combining character sequence is not part of a grapheme cluster and is subject to the same kinds of fallback processing as for any defective combining character sequence.

- Graphic application concerns visual rendering issues and thus is an issue for nonspacing marks that have visible glyphs. Those glyphs interact, in rendering, with their grapheme base.

Throughout the text of the standard, whenever the situation is clear, discussion of combining marks often simply talks about combining marks "applying" to their base. In the prototypical case of a nonspacing accent mark applying to a single base character letter, this simplification is not problematical, because the nonspacing mark both depends (notionally) on its base character and simultaneously applies (graphically) to its grapheme base, affecting its display. The finer distinctions are needed when dealing with the edge cases, such as combining marks that have no display glyph, graphical application of nonspacing marks to Korean syllables, and the behavior of spacing combining marks.

The distinction made here between notional dependence and graphical application of combining marks does not preclude spacing marks or even sequences of base characters from having effects on neighboring characters in rendering. Such effects on rendering are generally referred to in the Unicode Standard as *conjoining behavior*, and the details are typically script-specific. Thus spacing (or non-spacing) forms of dependent vowels (*matras*) in Indic scripts may trigger particular kinds of *conjunct formation* or may be repositioned in ways that influence the rendering of other characters. (See *Chapter 12, South and Central Asia-I*, for many examples.) In numerous scripts, sequences of base characters may form ligatures in rendering. (See *Section 23.2.2, Cursive Connection and Ligatures*.) In other scripts, sequences of base characters may be systematically organized into *syllable blocks* for display. (For Hangul, see *Section 3.12, Conjoining Jamo Behavior*. For the Khitan Small Script, see the description in *Section 18.12, Khitan Small Script*.)

The following listing specifies the principles regarding application of combining marks. Many of these principles are illustrated in *Section 2.11, Combining Characters*, and *Section 7.9, Combining Marks*.

P1 [Normative] Combining character order: Combining characters follow the base character on which they depend.

- This principle follows from the definition of a combining character sequence.

- Thus the character sequence <U+0061 "a" LATIN SMALL LETTER A, U+0308 "◌̈" COMBINING DIAERESIS, U+0075 "u" LATIN SMALL LETTER U> is unambiguously interpreted (and displayed) as "äu", not "aü". See *Figure 2-18*.

P2 [Guideline] Inside-out application. Nonspacing marks with the same combining class and spacing marks rendered on the same side of the base

are generally positioned graphically outward from the grapheme base to which they apply.

- The most numerous and important instances of this principle involve nonspacing marks applied either directly above or below a grapheme base. See *Figure 2-21*.

- In a sequence of two nonspacing marks above a grapheme base, the first nonspacing mark is placed directly above the grapheme base, and the second is then placed above the first nonspacing mark.

- In a sequence of two nonspacing marks below a grapheme base, the first nonspacing mark is placed directly below the grapheme base, and the second is then placed below the first nonspacing mark.

- In a sequence of two spacing marks rendered to the left of a grapheme base, the first spacing mark is placed directly to the left of the grapheme base, and the second is then placed to the left of the first spacing mark.

- In a sequence of two spacing marks rendered to the right of a grapheme base, the first spacing mark is placed directly to the right of the grapheme base, and the second is then placed to the right of the first spacing mark.

- This rendering behavior for nonspacing marks can be generalized to sequences of any length, although practical considerations usually limit such sequences to no more than two or three marks above and/or below a grapheme base.

- When applied to nonspacing marks, the principle of inside-out application is also referred to as *default stacking behavior* for nonspacing marks.

P3 [Guideline] Side-by-side application. Notwithstanding the principle of inside-out application, some specific nonspacing marks may override the default stacking behavior and are positioned side-by-side over (or under) a grapheme base, rather than stacking vertically.

- Such side-by-side positioning may reflect language-specific orthographic rules, such as for Vietnamese diacritics and tone marks or for polytonic Greek breathing and accent marks. See *Table 2-6*.

- Side-by-side positioning may also reflect certain writing conventions, such as for titlo letters in the Old Church Slavonic manuscript tradition.

- When positioned side-by-side, the visual rendering order of a sequence of nonspacing marks reflects the dominant order of the script with which they are used. Thus, in Greek, the first nonspacing mark in such a sequence will be positioned to the left side above a grapheme base, and the second to the right side above the grapheme base. In Hebrew, the opposite positioning is used for side-by-side placement.

- The combining parentheses diacritical marks U+1ABB..U+1ABD are also positioned in a side-by-side manner, surrounding other diacritics, as described in the subsection "Combining Diacritical Marks Extended: U+1AB0–U+1AFF" in *Section 7.9, Combining Marks*.

P4 [Guideline] Traditional typographical behavior will sometimes override the default placement or rendering of nonspacing marks.

- Because of typographical conflict with the descender of a base character, a combining comma below placed on a lowercase "g" is traditionally rendered as if it were an inverted comma above. See *Figure 7-1*.

- Because of typographical conflict with the ascender of a base character, a combining háček (caron) is traditionally rendered as an apostrophe when placed, for example, on a lowercase "d". See *Figure 7-1*.

- The relative placement of vowel marks in Arabic cannot be predicted by default stacking behavior alone, but depends on traditional rules of Arabic typography. See *Figure 9-5*.

P5 [Normative] Nondistinct order. Nonspacing marks with different, non-zero combining classes may occur in different orders without affecting either the visual display of a combining character sequence or the interpretation of that sequence.

- For example, if one nonspacing mark occurs above a grapheme base and another nonspacing mark occurs below it, they will have distinct combining classes. The order in which they occur in the combining character sequence does not matter for the display or interpretation of the resulting grapheme cluster.

- The introduction of the combining class for characters and its use in canonical ordering in the standard is to precisely define canonical equivalence and thereby clarify exactly which such alternate sequences must be considered as identical for display and interpretation. See *Figure 2-24*.

- In cases of nondistinct order, the order of combining marks has no linguistic significance. The order does not reflect how "closely bound" they are to the base. After canonical reordering, the order may no longer reflect the typed-in sequence. Rendering systems should be prepared to deal with common typed-in sequences and with canonically reordered sequences. See *Table 5-3*.

- Inserting a *combining grapheme joiner* between two combining marks with nondistinct order prevents their canonical reordering. For more information, see "Combining Grapheme Joiner" in *Section 23.2, Layout Controls*.

P6 [Guideline] *Enclosing marks surround their grapheme base and any intervening nonspacing marks.*

- This implies that enclosing marks successively surround previous enclosing marks. See *Figure 3-1*.

Figure 3-1. Enclosing Marks
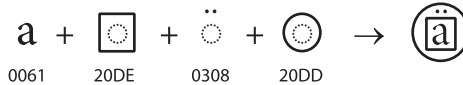
- Dynamic application of enclosing marks—particularly sequences of enclosing marks—is beyond the capability of most fonts and simple rendering processes. It is not unexpected to find fallback rendering in cases such as that illustrated in *Figure 3-1*.

P7 [Guideline] *Double diacritic nonspacing marks, such as* U+0360 COMBINING DOUBLE TILDE*, apply to their grapheme base, but are intended to be rendered with glyphs that encompass a following grapheme base as well.*

- Because such double diacritic display spans combinations of elements that would otherwise be considered grapheme clusters, the support of double diacritics in rendering may involve special handling for cursor placement and text selection. See *Figure 7-9* for an example.

P8 [Guideline] *When double diacritic nonspacing marks interact with normal nonspacing marks in a grapheme cluster, they "float" to the outermost layer of the stack of rendered marks (either above or below).*

- This behavior can be conceived of as a kind of looser binding of such double diacritics to their bases. In effect, all other nonspacing marks are applied first, and then the double diacritic will span the resulting stacks. See *Figure 7-10* for an example.

- Double diacritic nonspacing marks are also given a very high combining class, so that in canonical order they appear at or near the end of any combining character sequence. *Figure 7-11* shows an example of the use of CGJ to block this reordering.

- The interaction of enclosing marks and double diacritics is not well defined graphically. Many fonts and rendering processes may not be able to handle combinations of these marks. It is not recommended to use combinations of these together in the same grapheme cluster.

P9 [Guideline] *When a nonspacing mark above (a combining mark with ccc = 230) is applied to the letters i and j or any other character with the*

> **Soft_Dotted property, the inherent dot on the base character is suppressed in display.**

- See *Figure 7-2* for an example.

- For languages such as Lithuanian, in which both a dot and an accent must be displayed, use U+0307 COMBINING DOT ABOVE. For guidelines in handling this situation in case mapping, see *Section 5.18, Case Mappings*.

Combining Marks and Korean Syllables. When a grapheme cluster comprises a Korean syllable, a combining mark applies to that entire syllable. For example, in the following sequence the *double dot tone mark* is applied to the entire Korean syllable, not just to the last jamo:

U+1100 ㄱ *choseong kiyeok* + U+1161 ㅏ *jungseong a* + U+302F ◌ *double dot tone mark* → ꞉가

If the combining mark in question is an *enclosing* combining mark, then it would enclose the entire Korean syllable, rather than the last jamo in it:

U+1100 ㄱ *choseong kiyeok* + U+1161 ㅏ *jungseong a* + U+20DD ◎ *enclosing circle* → Ⓖ

This treatment of the application of combining marks with respect to Korean syllables follows from the implications of canonical equivalence. It should be noted, however, that older implementations may have supported the application of an enclosing combining mark to an entire Indic consonant conjunct or to a sequence of grapheme clusters linked together by combining grapheme joiners. Such an approach has a number of technical problems and leads to interoperability defects, so it is strongly recommended that implementations do not follow it.

For more information on the recommended use of the combining grapheme joiner, see the subsection "Combining Grapheme Joiner" in *Section 23.2, Layout Controls*. For more discussion regarding the application of combining marks in general, see *Section 7.9, Combining Marks*.

3.7 Decomposition

D62 *Decomposition mapping:* A mapping from a character to a sequence of one or
more characters that is a canonical or compatibility equivalent, and that is
listed in the character names list or described in *Section 3.12, Conjoining Jamo
Behavior*.

- Each character has at most one decomposition mapping. The mappings in
Section 3.12, Conjoining Jamo Behavior, are canonical mappings. The mappings
in the character names list are identified as either canonical or compatibility
mappings (see *Section 24.1, Character Names List*).

D63 *Decomposable character:* A character that is equivalent to a sequence of one or
more other characters, according to the decomposition mappings found in the
Unicode Character Database, and those described in *Section 3.12, Conjoining
Jamo Behavior*.

- A decomposable character is also referred to as a *precomposed* character or
composite character.
- The decomposition mappings from the Unicode Character Database are also
given in *Section 24.1, Character Names List*.

D64 *Decomposition:* A sequence of one or more characters that is equivalent to a
decomposable character. A full decomposition of a character sequence results
from decomposing each of the characters in the sequence until no characters
can be further decomposed.

3.7.1 Compatibility Decomposition

D65 *Compatibility decomposition:* The decomposition of a character or character
sequence that results from recursively applying *both* the compatibility
mappings *and* the canonical mappings found in the Unicode Character Data-
base, and those described in *Section 3.12, Conjoining Jamo Behavior*, until no
characters can be further decomposed, and then reordering nonspacing marks
according to *Section 3.11, Normalization Forms*.

- The decomposition mappings from the Unicode Character Database are also
given in *Section 24.1, Character Names List*.
- Some compatibility decompositions remove formatting information.

D66 *Compatibility decomposable character:* A character whose compatibility decom-
position is not identical to its canonical decomposition. It may also be known

as a *compatibility precomposed* character or a *compatibility composite* character.

- For example, U+00B5 MICRO SIGN has no canonical decomposition mapping, so its canonical decomposition is the same as the character itself. It has a compatibility decomposition to U+03BC GREEK SMALL LETTER MU. Because MICRO SIGN has a compatibility decomposition that is not equal to its canonical decomposition, it is a compatibility decomposable character.

- For example, U+03D3 GREEK UPSILON WITH ACUTE AND HOOK SYMBOL canonically decomposes to the sequence <U+03D2 GREEK UPSILON WITH HOOK SYMBOL, U+0301 COMBINING ACUTE ACCENT>. That sequence has a compatibility decomposition of <U+03A5 GREEK CAPITAL LETTER UPSILON, U+0301 COMBINING ACUTE ACCENT>. Because GREEK UPSILON WITH ACUTE AND HOOK SYMBOL has a compatibility decomposition that is not equal to its canonical decomposition, it is a compatibility decomposable character.

- This term should not be confused with the term "compatibility character," which is discussed in *Section 2.3, Compatibility Characters*.

- Many compatibility decomposable characters are included in the Unicode Standard solely to represent distinctions in other base standards. They support transmission and processing of legacy data. Their use is discouraged other than for legacy data or other special circumstances.

- Some widely used and indispensable characters, such as NBSP, are compatibility decomposable characters for historical reasons. Their use is not discouraged.

- A large number of compatibility decomposable characters are used in phonetic and mathematical notation, where their use is not discouraged.

- For historical reasons, some characters that might have been given a compatibility decomposition were not, in fact, decomposed. The Normalization Stability Policy prohibits adding decompositions for such cases in the future, so that normalization forms will stay stable. See the subsection "Policies" in *Section B.3, Other Unicode Online Resources*.

- Replacing a compatibility decomposable character by its compatibility decomposition may lose round-trip convertibility with a base standard.

D67 *Compatibility equivalent:* Two character sequences are said to be compatibility equivalents if their full compatibility decompositions are identical.

3.7.2 *Canonical Decomposition*

D68 *Canonical decomposition:* The decomposition of a character or character sequence that results from recursively applying the canonical mappings found in the Unicode Character Database and those described in *Section 3.12, Conjoining Jamo Behavior*, until no characters can be further decomposed, and

then reordering nonspacing marks according to *Section 3.11, Normalization Forms*.

- The decomposition mappings from the Unicode Character Database are also printed in *Section 24.1, Character Names List*.

- A canonical decomposition does not remove formatting information.

D69 *Canonical decomposable character:* A character that is not identical to its canonical decomposition. It may also be known as a *canonical precomposed* character or a *canonical composite* character.

- For example, U+00E0 LATIN SMALL LETTER A WITH GRAVE is a canonical decomposable character because its canonical decomposition is to the sequence <U+0061 LATIN SMALL LETTER A, U+0300 COMBINING GRAVE ACCENT>. U+212A KELVIN SIGN is a canonical decomposable character because its canonical decomposition is to U+004B LATIN CAPITAL LETTER K.

D70 *Canonical equivalent:* Two character sequences are said to be canonical equivalents if their full canonical decompositions are identical.

- For example, the sequences <*o, combining-diaeresis*> and <*ö*> are canonical equivalents. Canonical equivalence is a Unicode property. It should not be confused with language-specific collation or matching, which may add other equivalencies. For example, in Swedish, *ö* is treated as a completely different letter from *o* and is collated after *z*. In German, *ö* is weakly equivalent to *oe* and is collated with *oe*. In English, *ö* is just an *o* with a diacritic that indicates that it is pronounced separately from the previous letter (as in *coöperate*) and is collated with *o*.

- By definition, all canonical-equivalent sequences are also compatibility-equivalent sequences.

For information on the use of decomposition in normalization, see *Section 3.11, Normalization Forms*.

3.8 Surrogates

D71 *High-surrogate code point:* A Unicode code point in the range U+D800 to U+DBFF.

D72 *High-surrogate code unit:* A 16-bit code unit in the range $D800_{16}$ to $DBFF_{16}$, used in UTF-16 as the leading code unit of a surrogate pair.

D73 *Low-surrogate code point:* A Unicode code point in the range U+DC00 to U+DFFF.

D74 *Low-surrogate code unit:* A 16-bit code unit in the range $DC00_{16}$ to $DFFF_{16}$, used in UTF-16 as the trailing code unit of a surrogate pair.

• High-surrogate and low-surrogate code points are designated only for that use.

• High-surrogate and low-surrogate code units are used *only* in the context of the UTF-16 character encoding form.

D75 *Surrogate pair:* A representation for a single abstract character that consists of a sequence of two 16-bit code units, where the first value of the pair is a high-surrogate code unit and the second value is a low-surrogate code unit.

• Surrogate pairs are used only in UTF-16. (See *Section 3.9, Unicode Encoding Forms*.)

• Isolated surrogate code units have no interpretation on their own. Certain other isolated code units in other encoding forms also have no interpretation on their own. For example, the isolated byte 80_{16} has no interpretation in UTF-8; it can be used *only* as part of a multibyte sequence. (See *Table 3-7*.)

• Sometimes high-surrogate code units are referred to as *leading surrogates*. Low-surrogate code units are then referred to as *trailing surrogates*. This is analogous to usage in UTF-8, which has *leading bytes* and *trailing bytes*.

• For more information, see *Section 23.6, Surrogates Area*, and *Section 5.4, Handling Surrogate Pairs in UTF-16*.

3.9 Unicode Encoding Forms

The Unicode Standard supports three character encoding forms: UTF-32, UTF-16, and UTF-8. Each encoding form maps the Unicode code points U+0000..U+D7FF and U+E000..U+10FFFF to unique code unit sequences. The size of the code unit is specified for each encoding form. This section presents the formal definition of each of these encoding forms.

D76 *Unicode scalar value:* Any Unicode code point except high-surrogate and low-surrogate code points.

- As a result of this definition, the set of Unicode scalar values consists of the ranges 0 to $D7FF_{16}$ and $E000_{16}$ to $10FFFF_{16}$, inclusive.

D77 *Code unit:* The minimal bit combination that can represent a unit of encoded text for processing or interchange.

- Code units are particular units of computer storage. Other character encoding standards typically use code units defined as 8-bit units—that is, *octets*. The Unicode Standard uses 8-bit code units in the UTF-8 encoding form, 16-bit code units in the UTF-16 encoding form, and 32-bit code units in the UTF-32 encoding form.

- A code unit is also referred to as a *code value* in the information industry.

- In the Unicode Standard, specific values of some code units cannot be used to represent an encoded character in isolation. This restriction applies to isolated surrogate code units in UTF-16 and to the bytes 80–FF in UTF-8. Similar restrictions apply for the implementations of other character encoding standards; for example, the bytes 81–9F, E0–FC in SJIS (Shift-JIS) cannot represent an encoded character by themselves.

- For information on use of wchar_t or other programming language types to represent Unicode code units, see "ANSI/ISO C wchar_t" in <u>Section 5.2, Programming Languages and Data Types</u>.

D78 *Code unit sequence:* An ordered sequence of one or more code units.

- When the code unit is an 8-bit unit, a code unit sequence may also be referred to as a *byte sequence*.

- A code unit sequence may consist of a single code unit.

- In the context of programming languages, the *value* of a *string* data type basically consists of a code unit sequence. Informally, a code unit sequence is itself just referred to as a *string*, and a *byte sequence* is referred to as a *byte string*. Care must be taken in making this terminological equivalence, however,

because the formally defined concept of a string may have additional requirements or complications in programming languages. For example, a *string* is defined as a *pointer to char* in the C language and is conventionally terminated with a NULL character. In object-oriented languages, a *string* is a complex object, with associated methods, and its value may or may not consist of merely a code unit sequence.

- Depending on the structure of a character encoding standard, it may be necessary to use a code unit sequence (of more than one unit) to represent a single encoded character. For example, the code unit in SJIS is a byte: encoded characters such as "a" can be represented with a single byte in SJIS, whereas ideographs require a sequence of two code units. The Unicode Standard also makes use of code unit sequences whose length is greater than one code unit.

D79 *Unicode encoding form:* A mapping from each Unicode scalar value to a unique code unit sequence.

- This standard defines three Unicode encoding forms. See D90, D91, and D92.

- Unless otherwise stated, the term *Unicode encoding form* refers to one of those three forms. For clarity, they can be referred to as *standard Unicode encoding forms*.

- For historical reasons, the Unicode encoding forms are also referred to as *Unicode* (or *UCS*) *transformation formats* (UTF). That term is actually ambiguous between its usage for encoding forms and encoding schemes.

- The mapping of the set of Unicode scalar values to the set of code unit sequences for a Unicode encoding form is *one-to-one*. This property guarantees that a reverse mapping can always be derived. Given the mapping of any Unicode scalar value to a particular code unit sequence for a given encoding form, one can derive the original Unicode scalar value unambiguously from that code unit sequence.

- The mapping of the set of Unicode scalar values to the set of code unit sequences for a Unicode encoding form is not *onto*. In other words, for any given encoding form, there exist code unit sequences that have no associated Unicode scalar value.

- To ensure that the mapping for a Unicode encoding form is one-to-one, *all* Unicode scalar values, including those corresponding to noncharacter code points and unassigned code points, must be mapped to unique code unit sequences. Note that this requirement does not extend to high-surrogate and low-surrogate code points, which are excluded by definition from the set of Unicode scalar values.

D80 *Unicode string:* A code unit sequence containing code units of a particular Unicode encoding form.

- In the rawest form, Unicode strings may be implemented simply as arrays of the appropriate integral data type, consisting of a sequence of code units lined up one immediately after the other.

- A single Unicode string must contain only code units from a single Unicode encoding form. It is not permissible to mix forms within a string.

D81 *Unicode 8-bit string:* A Unicode string containing only UTF-8 code units.

D82 *Unicode 16-bit string:* A Unicode string containing only UTF-16 code units.

D83 *Unicode 32-bit string:* A Unicode string containing only UTF-32 code units.

D84 *Ill-formed:* A Unicode code unit sequence that purports to be in a Unicode encoding form is called *ill-formed* if and only if it does *not* follow the specification of that Unicode encoding form.

- Any code unit sequence that would correspond to a code point outside the defined range of Unicode scalar values would, for example, be ill-formed.

- UTF-8 has some strong constraints on the possible byte ranges for leading and trailing bytes. A violation of those constraints would produce a code unit sequence that could not be mapped to a Unicode scalar value, resulting in an ill-formed code unit sequence.

D84a *Ill-formed code unit subsequence:* A non-empty subsequence of a Unicode code unit sequence X which does not contain any code units which also belong to any minimal well-formed subsequence of X.

- In other words, an ill-formed code unit subsequence cannot overlap with a minimal well-formed subsequence.

D85 *Well-formed:* A Unicode code unit sequence that purports to be in a Unicode encoding form is called *well-formed* if and only if it *does* follow the specification of that Unicode encoding form.

D85a *Minimal well-formed code unit subsequence:* A well-formed Unicode code unit sequence that maps to a single Unicode scalar value.

- For UTF-8, see the specification in D92 and <u>Table 3-7</u>.

- For UTF-16, see the specification in D91.

- For UTF-32, see the specification in D90.

A well-formed Unicode code unit sequence can be partitioned into one or more minimal well-formed code unit sequences for the given Unicode encoding form. Any Unicode code unit sequence can be partitioned into subsequences that are either well-formed or ill-formed. The sequence as a whole is well-formed if and only if it contains no ill-formed subsequence. The sequence as a whole is ill-formed if and only if it contains at least one ill-formed subsequence.

D86 *Well-formed UTF-8 code unit sequence:* A well-formed Unicode code unit sequence of UTF-8 code units.

- The UTF-8 code unit sequence <41 C3 B1 42> is well-formed, because it can be partitioned into subsequences, all of which match the specification for UTF-8 in *Table 3-7*. It consists of the following minimal well-formed code unit subsequences: <41>, <C3 B1>, and <42>.

- The UTF-8 code unit sequence <41 C2 C3 B1 42> is ill-formed, because it contains one ill-formed subsequence. There is no subsequence for the C2 byte which matches the specification for UTF-8 in *Table 3-7*. The code unit sequence is partitioned into one minimal well-formed code unit subsequence, <41>, followed by one ill-formed code unit subsequence, <C2>, followed by two minimal well-formed code unit subsequences, <C3 B1> and <42>.

- In isolation, the UTF-8 code unit sequence <C2 C3> would be ill-formed, but in the context of the UTF-8 code unit sequence <41 C2 C3 B1 42>, <C2 C3> does not constitute an ill-formed code unit subsequence, because the C3 byte is actually the first byte of the minimal well-formed UTF-8 code unit subsequence <C3 B1>. Ill-formed code unit subsequences do not overlap with minimal well-formed code unit subsequences.

D87 *Well-formed UTF-16 code unit sequence:* A well-formed Unicode code unit sequence of UTF-16 code units.

D88 *Well-formed UTF-32 code unit sequence:* A well-formed Unicode code unit sequence of UTF-32 code units.

D89 *In a Unicode encoding form:* A Unicode string is said to be *in* a particular Unicode encoding form if and only if it consists of a well-formed Unicode code unit sequence of that Unicode encoding form.

- A Unicode string consisting of a well-formed UTF-8 code unit sequence is said to be *in UTF-8*. Such a Unicode string is referred to as a *valid UTF-8 string*, or a *UTF-8 string* for short.

- A Unicode string consisting of a well-formed UTF-16 code unit sequence is said to be *in UTF-16*. Such a Unicode string is referred to as a *valid UTF-16 string*, or a *UTF-16 string* for short.

- A Unicode string consisting of a well-formed UTF-32 code unit sequence is said to be *in UTF-32*. Such a Unicode string is referred to as a *valid UTF-32 string*, or a *UTF-32 string* for short.

Unicode strings need not contain well-formed code unit sequences under all conditions. This is equivalent to saying that a particular Unicode string need not be *in* a Unicode encoding form.

- For example, it is perfectly reasonable to talk about an operation that takes the two Unicode 16-bit strings, <004D D800> and <DF02 004D>, each of which contains an ill-formed UTF-16 code unit sequence, and concatenates them to form another Unicode string <004D D800 DF02 004D>, which contains a well-formed UTF-16 code unit sequence. The first two Unicode strings are not *in* UTF-16, but the resultant Unicode string is.

- As another example, the code unit sequence <C0 80 61 F3> is a Unicode 8-bit string, but does not consist of a well-formed UTF-8 code unit sequence. That code unit sequence could not result from the specification of the UTF-8 encoding form and is thus ill-formed. (The same code unit sequence could, of course, be well-formed in the context of some other character encoding standard using 8-bit code units, such as ISO/IEC 8859-1, or vendor code pages.)

If a Unicode string *purports* to be *in* a Unicode encoding form, then it must not contain any ill-formed code unit subsequence.

If a process which verifies that a Unicode string is in a Unicode encoding form encounters an ill-formed code unit subsequence in that string, then it must not identify that string as being in that Unicode encoding form.

A process which interprets a Unicode string must not interpret any ill-formed code unit subsequences in the string as characters. (See conformance clause C10.) Furthermore, such a process must not treat any adjacent well-formed code unit sequences as being part of those ill-formed code unit sequences.

Table 3-4 gives examples that summarize the three Unicode encoding forms.

Table 3-4. Examples of Unicode Encoding Forms

Code Point	Encoding Form	Code Unit Sequence
U+004D	UTF-32	0000004D
	UTF-16	004D
	UTF-8	4D
U+0430	UTF-32	00000430
	UTF-16	0430
	UTF-8	D0 B0

U+4E8C	UTF-32	00004E8C
	UTF-16	4E8C
	UTF-8	E4 BA 8C
U+10302	UTF-32	00010302
	UTF-16	D800 DF02
	UTF-8	F0 90 8C 82

3.9.1 UTF-32

D90 *UTF-32 encoding form:* The Unicode encoding form that assigns each Unicode scalar value to a single unsigned 32-bit code unit with the same numeric value as the Unicode scalar value.

- In UTF-32, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <0000004D 00000430 00004E8C 00010302>.

- Because surrogate code points are not included in the set of Unicode scalar values, UTF-32 code units in the range $0000D800_{16}$..$0000DFFF_{16}$ are ill-formed.

- Any UTF-32 code unit greater than $0010FFFF_{16}$ is ill-formed.

For a discussion of the relationship between UTF-32 and UCS-4 encoding form defined in ISO/IEC 10646, see *Appendix C.2, Encoding Forms in ISO/IEC 10646*.

3.9.2 UTF-16

D91 *UTF-16 encoding form:* The Unicode encoding form that assigns each Unicode scalar value in the ranges U+0000..U+D7FF and U+E000..U+FFFF to a single unsigned 16-bit code unit with the same numeric value as the Unicode scalar value, and that assigns each Unicode scalar value in the range U+10000..U+10FFFF to a surrogate pair, according to *Table 3-5*.

- In UTF-16, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <004D 0430 4E8C D800 DF02>, where <D800 DF02> corresponds to U+10302.

- Because surrogate code points are not Unicode scalar values, isolated UTF-16 code units in the range $D800_{16}$..$DFFF_{16}$ are ill-formed.

Table 3-5 specifies the bit distribution for the UTF-16 encoding form. Note that for Unicode scalar values equal to or greater than U+10000, UTF-16 uses surrogate pairs. Calculation of the surrogate pair values involves subtraction of 10000_{16}, to account for the starting offset to the scalar value. ISO/IEC 10646 specifies an equivalent UTF-16 encoding form. For details, see *Appendix C.3, UTF-8 and UTF-16*.

Table 3-5. UTF-16 Bit Distribution

Scalar Value	UTF-16
xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
000uuuuuxxxxxxxxxxxxxxxx	110110wwwwxxxxxx 110111xxxxxxxxxx

Note: wwww = uuuuu - 1

3.9.3 UTF-8

D92 *UTF-8 encoding form:* The Unicode encoding form that assigns each Unicode scalar value to an unsigned byte sequence of one to four bytes in length, as specified in *Table 3-6* and *Table 3-7*.

• In UTF-8, the code point sequence <004D, 0430, 4E8C, 10302> is represented as <4D D0 B0 E4 BA 8C F0 90 8C 82>, where <4D> corresponds to U+004D, <D0 B0> corresponds to U+0430, <E4 BA 8C> corresponds to U+4E8C, and <F0 90 8C 82> corresponds to U+10302.

• Any UTF-8 byte sequence that does not match the patterns listed in *Table 3-7* is ill-formed.

• Before the Unicode Standard, Version 3.1, the problematic "non-shortest form" byte sequences in UTF-8 were those where BMP characters could be represented in more than one way. These sequences are ill-formed, because they are not allowed by *Table 3-7*.

• Because surrogate code points are not Unicode scalar values, any UTF-8 byte sequence that would otherwise map to code points U+D800..U+DFFF is ill-formed.

Table 3-6 specifies the bit distribution for the UTF-8 encoding form, showing the ranges of Unicode scalar values corresponding to one-, two-, three-, and four-byte sequences. For a discussion of the difference in the formulation of UTF-8 in ISO/IEC 10646, see *Appendix C.3, UTF-8 and UTF-16*.

Table 3-6. UTF-8 Bit Distribution

Scalar Value	First Byte	Second Byte	Third Byte	Fourth Byte
00000000 0xxxxxxx	0xxxxxxx			
00000yyy yyxxxxxx	110yyyyy	10xxxxxx		
zzzzyyyy yyxxxxxx	1110zzzz	10yyyyyy	10xxxxxx	
000uuuuu zzzzyyyy yyxxxxxx	11110uuu	10uuzzzz	10yyyyyy	10xxxxxx

Table 3-7 lists all of the byte sequences that are well-formed in UTF-8. A range of byte values such as A0..BF indicates that any byte from A0 to BF (inclusive) is well-

formed in that position. Any byte value outside of the ranges listed is ill-formed. For example:

- The byte sequence <C0 AF> is *ill-formed*, because C0 is not well-formed in the "First Byte" column.

- The byte sequence <E0 9F 80> is *ill-formed*, because in the row where E0 is well-formed as a first byte, 9F is not well-formed as a second byte.

- The byte sequence <F4 80 83 92> is *well-formed*, because every byte in that sequence matches a byte range in a row of the table (the last row).

Table 3-7. Well-Formed UTF-8 Byte Sequences

Code Points	First Byte	Second Byte	Third Byte	Fourth Byte
U+0000..U+007F	00..7F			
U+0080..U+07FF	C2..DF	80..BF		
U+0800..U+0FFF	E0	*A0*..BF	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80..*9F*	80..BF	
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	*90*..BF	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80..*8F*	80..BF	80..BF

In *Table 3-7*, cases where a trailing byte range is not 80..BF are shown in bold italic to draw attention to them. These exceptions to the general pattern occur only in the second byte of a sequence.

As a consequence of the well-formedness conditions specified in *Table 3-7*, the following byte values are disallowed in UTF-8: C0–C1, F5–FF.

3.9.4 Encoding Form Conversion

D93 *Encoding form conversion:* A conversion defined directly between the code unit sequences of one Unicode encoding form and the code unit sequences of another Unicode encoding form.

- In implementations of the Unicode Standard, a typical API will logically convert the input code unit sequence into Unicode scalar values (code points) and then convert those Unicode scalar values into the output code unit sequence. Proper analysis of the encoding forms makes it possible to convert the code units directly, thereby obtaining the same results but with a more efficient process.

- A conformant encoding form conversion will treat any ill-formed code unit sequence as an error condition. (See conformance clause C10.) This guarantees

that it will neither interpret nor emit an ill-formed code unit sequence. Any implementation of encoding form conversion must take this requirement into account, because an encoding form conversion implicitly involves a verification that the Unicode strings being converted do, in fact, contain well-formed code unit sequences.

3.9.5 Constraints on Conversion Processes

The requirement not to interpret any ill-formed code unit subsequences in a string as characters (see conformance clause C10) has important consequences for conversion processes. Such processes may, for example, interpret UTF-8 code unit sequences as Unicode character sequences. If the converter encounters an ill-formed UTF-8 code unit sequence which starts with a valid first byte, but which does not continue with valid successor bytes (see *Table 3-7*), it *must not* consume the successor bytes as part of the ill-formed subsequence whenever those successor bytes themselves constitute part of a well-formed UTF-8 code unit subsequence.

If an implementation of a UTF-8 conversion process stops at the first error encountered, without reporting the end of any ill-formed UTF-8 code unit subsequence, then the requirement makes little practical difference. However, the requirement does introduce a significant constraint if the UTF-8 converter continues past the point of a detected error, perhaps by substituting one or more U+FFFD replacement characters for the uninterpretable, ill-formed UTF-8 code unit subsequence. For example, with the input UTF-8 code unit sequence <C2 41 42>, such a UTF-8 conversion process must not return <U+FFFD> or <U+FFFD, U+0042>, because either of those outputs would be the result of misinterpreting a well-formed subsequence as being part of the ill-formed subsequence. The expected return value for such a process would instead be <U+FFFD, U+0041, U+0042>.

For a UTF-8 conversion process to consume valid successor bytes is not only non-conformant, but also leaves the converter open to security exploits. See Unicode Technical Report #36, "Unicode Security Considerations."

Although a UTF-8 conversion process is required to never consume well-formed subsequences as part of its error handling for ill-formed subsequences, such a process is not otherwise constrained in how it deals with any ill-formed subsequence itself. An ill-formed subsequence consisting of more than one code unit could be treated as a single error or as multiple errors.

For example, in processing the UTF-8 code unit sequence <F0 80 80 41>, the only formal requirement mandated by Unicode conformance for a converter is that the <41> be processed and correctly interpreted as <U+0041>. The converter could return <U+FFFD, U+0041>, handling <F0 80 80> as a single error, or <U+FFFD, U+FFFD, U+FFFD, U+0041>, handling each byte of <F0 80 80> as a separate error,

or could take other approaches to signalling <F0 80 80> as an ill-formed code unit subsequence.

3.9.6 U+FFFD Substitution of Maximal Subparts

An increasing number of implementations are adopting the handling of ill-formed subsequences as specified in the W3C standard for encoding to achieve consistent U+FFFD replacements. See:

> http://www.w3.org/TR/encoding/

Although the Unicode Standard does not require this practice for conformance, the following text describes this practice and gives detailed examples.

D93a *Unconvertible offset:* An offset in a code unit sequence for which no code unit subsequence starting at that offset is well-formed.

D93b *Maximal subpart of an ill-formed subsequence:* The longest code unit subsequence starting at an unconvertible offset that is either:

a. the initial subsequence of a well-formed code unit sequence, or

b. a subsequence of length one.

This definition of the maximal subpart is used in describing how far to advance processing when making substitutions: always process at least one code unit, or as many code units as match the beginning of a well-formed character, up to the point where the next code unit would make it ill-formed, that is, an offset is reached that does not continue this partial character.

Or stated more formally:

> **Whenever an unconvertible offset is reached during conversion of a code unit sequence:**
>
> **1. The maximal subpart at that offset is replaced by a single U+FFFD.**
>
> **2. The conversion proceeds at the offset immediately after the maximal subpart.**

This practice of substituting maximal subparts can be trivially applied to the UTF-32 or UTF-16 encoding forms, but is primarily of interest when converting UTF-8 strings.

Unless the beginning of an ill-formed subsequence matches the beginning of some well-formed sequence, this practice replaces almost every byte of an ill-formed UTF-8 sequence with one U+FFFD. For example, every byte of a "non-shortest form"

sequence (see Definition D92), or of a truncated version thereof, is replaced, as shown in *Table 3-8*. (The interpretation of "non-shortest form" sequences has been forbidden since the publication of Corrigendum #1.)

Table 3-8. U+FFFD for Non-Shortest Form Sequences

Bytes	C0	AF	E0	80	BF	F0	81	82	41
Output	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	0041

Also, every byte of a sequence that would correspond to a surrogate code point, or of a truncated version thereof, is replaced with one U+FFFD, as shown in *Table 3-9*. (The interpretation of such byte sequences has been forbidden since Unicode 3.2.)

Table 3-9. U+FFFD for Ill-Formed Sequences for Surrogates

Bytes	ED	A0	80	ED	BF	BF	ED	AF	41
Output	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	0041

Finally, every byte of a sequence that would correspond to a code point beyond U+10FFFF, and any other byte that does not contribute to a valid sequence, is also replaced with one U+FFFD, as shown in *Table 3-10*.

Table 3-10. U+FFFD for Other Ill-Formed Sequences

Bytes	F4	91	92	93	FF	41	80	BF	42
Output	FFFD	FFFD	FFFD	FFFD	FFFD	0041	FFFD	FFFD	0042

Only when a sequence of two or three bytes is a truncated version of a sequence which is otherwise well-formed to that point, is more than one byte replaced with a single U+FFFD, as shown in *Table 3-11*.

Table 3-11. U+FFFD for Truncated Sequences

Bytes	E1	80	E2		F0	91	92	F1	BF	41
Output	FFFD			FFFD		FFFD			FFFD	0041

For a discussion of the generalization of this approach for conversion of other character sets to Unicode, see *Section 5.22, U+FFFD Substitution in Conversion*.

3.10 Unicode Encoding Schemes

D94   *Unicode encoding scheme:* A specified byte serialization for a Unicode encoding form, including the specification of the handling of a byte order mark (BOM), if allowed.

• For historical reasons, the Unicode encoding schemes are also referred to as *Unicode* (or *UCS*) *transformation formats* (UTF). That term is, however, ambiguous between its usage for encoding forms and encoding schemes.

The Unicode Standard supports seven encoding schemes. This section presents the formal definition of each of these encoding schemes.

D95   *UTF-8 encoding scheme:* The Unicode encoding scheme that serializes a UTF-8 code unit sequence in exactly the same order as the code unit sequence itself.

• In the UTF-8 encoding scheme, the UTF-8 code unit sequence <4D D0 B0 E4 BA 8C F0 90 8C 82> is serialized as <4D D0 B0 E4 BA 8C F0 90 8C 82>.

• Because the UTF-8 encoding form already deals in ordered byte sequences, the UTF-8 encoding scheme is trivial. The byte ordering is already obvious and completely defined by the UTF-8 code unit sequence itself. The UTF-8 encoding scheme is defined merely for completeness of the Unicode character encoding model.

• While there is obviously no need for a byte order signature when using UTF-8, there are occasions when processes convert UTF-16 or UTF-32 data containing a byte order mark into UTF-8. When represented in UTF-8, the byte order mark turns into the byte sequence <EF BB BF>. Its usage at the beginning of a UTF-8 data stream is not required by the Unicode Standard, but its presence does not affect conformance to the UTF-8 encoding scheme. Identification of the <EF BB BF> byte sequence at the beginning of a data stream can, however, be taken as a near-certain indication that the data stream is using the UTF-8 encoding scheme.

D96   *UTF-16BE encoding scheme:* The Unicode encoding scheme that serializes a UTF-16 code unit sequence as a byte sequence in big-endian format.

• In UTF-16BE, the UTF-16 code unit sequence <004D 0430 4E8C D800 DF02> is serialized as <00 4D 04 30 4E 8C D8 00 DF 02>.

• In UTF-16BE, an initial byte sequence <FE FF> is interpreted as U+FEFF ZERO WIDTH NO-BREAK SPACE.

D97 *UTF-16LE encoding scheme:* The Unicode encoding scheme that serializes a UTF-16 code unit sequence as a byte sequence in little-endian format.

- In UTF-16LE, the UTF-16 code unit sequence <004D 0430 4E8C D800 DF02> is serialized as <4D 00 30 04 8C 4E 00 D8 02 DF>.

- In UTF-16LE, an initial byte sequence <FF FE> is interpreted as U+FEFF ZERO WIDTH NO-BREAK SPACE.

D98 *UTF-16 encoding scheme:* The Unicode encoding scheme that serializes a UTF-16 code unit sequence as a byte sequence in either big-endian or little-endian format.

- In the UTF-16 encoding scheme, the UTF-16 code unit sequence <004D 0430 4E8C D800 DF02> is serialized as <FE FF 00 4D 04 30 4E 8C D8 00 DF 02> or <FF FE 4D 00 30 04 8C 4E 00 D8 02 DF> or <00 4D 04 30 4E 8C D8 00 DF 02>.

- In the UTF-16 encoding scheme, an initial byte sequence corresponding to U+FEFF is interpreted as a byte order mark; it is used to distinguish between the two byte orders. An initial byte sequence <FE FF> indicates big-endian order, and an initial byte sequence <FF FE> indicates little-endian order. The BOM is not considered part of the content of the text.

- The UTF-16 encoding scheme may or may not begin with a BOM. However, when there is no BOM, and in the absence of a higher-level protocol, the byte order of the UTF-16 encoding scheme is big-endian.

Table 3-12 gives examples that summarize the three Unicode encoding schemes for the UTF-16 encoding form.

Table 3-12. Summary of UTF-16BE, UTF-16LE, and UTF-16

Code Unit Sequence	Encoding Scheme	Byte Sequence(s)
004D	UTF-16BE	00 4D
	UTF-16LE	4D 00
	UTF-16	FE FF 00 4D FF FE 4D 00 00 4D
0430	UTF-16BE	04 30
	UTF-16LE	30 04
	UTF-16	FE FF 04 30 FF FE 30 04 04 30
4E8C	UTF-16BE	4E 8C
	UTF-16LE	8C 4E

		FE FF 4E 8C FF FE 8C 4E 4E 8C
	UTF-16	
D800 DF02	UTF-16BE	D8 00 DF 02
	UTF-16LE	00 D8 02 DF
	UTF-16	FE FF D8 00 DF 02 FF FE 00 D8 02 DF D8 00 DF 02

D99 *UTF-32BE encoding scheme:* The Unicode encoding scheme that serializes a UTF-32 code unit sequence as a byte sequence in big-endian format.

- In UTF-32BE, the UTF-32 code unit sequence <0000004D 00000430 00004E8C 00010302> is serialized as <00 00 00 4D 00 00 04 30 00 00 4E 8C 00 01 03 02>.

- In UTF-32BE, an initial byte sequence <00 00 FE FF> is interpreted as U+FEFF ZERO WIDTH NO-BREAK SPACE.

D100 *UTF-32LE encoding scheme:* The Unicode encoding scheme that serializes a UTF-32 code unit sequence as a byte sequence in little-endian format.

- In UTF-32LE, the UTF-32 code unit sequence <0000004D 00000430 00004E8C 00010302> is serialized as <4D 00 00 00 30 04 00 00 8C 4E 00 00 02 03 01 00>.

- In UTF-32LE, an initial byte sequence <FF FE 00 00> is interpreted as U+FEFF ZERO WIDTH NO-BREAK SPACE.

D101 *UTF-32 encoding scheme:* The Unicode encoding scheme that serializes a UTF-32 code unit sequence as a byte sequence in either big-endian or little-endian format.

- In the UTF-32 encoding scheme, the UTF-32 code unit sequence <0000004D 00000430 00004E8C 00010302> is serialized as <00 00 FE FF 00 00 00 4D 00 00 04 30 00 00 4E 8C 00 01 03 02> or <FF FE 00 00 4D 00 00 00 30 04 00 00 8C 4E 00 00 02 03 01 00> or <00 00 00 4D 00 00 04 30 00 00 4E 8C 00 01 03 02>.

- In the UTF-32 encoding scheme, an initial byte sequence corresponding to U+FEFF is interpreted as a byte order mark; it is used to distinguish between the two byte orders. An initial byte sequence <00 00 FE FF> indicates big-endian order, and an initial byte sequence <FF FE 00 00> indicates little-endian order. The BOM is not considered part of the content of the text.

- The UTF-32 encoding scheme may or may not begin with a BOM. However, when there is no BOM, and in the absence of a higher-level protocol, the byte order of the UTF-32 encoding scheme is big-endian.

Table 3-13 gives examples that summarize the three Unicode encoding schemes for the UTF-32 encoding form.

Table 3-13. Summary of UTF-32BE, UTF-32LE, and UTF-32

Code Unit Sequence	Encoding Scheme	Byte Sequence(s)
0000004D	UTF-32BE	00 00 00 4D
	UTF-32LE	4D 00 00 00
	UTF-32	00 00 FE FF 00 00 00 4D FF FE 00 00 4D 00 00 00 00 00 00 4D
00000430	UTF-32BE	00 00 04 30
	UTF-32LE	30 04 00 00
	UTF-32	00 00 FE FF 00 00 04 30 FF FE 00 00 30 04 00 00 00 00 04 30
00004E8C	UTF-32BE	00 00 4E 8C
	UTF-32LE	8C 4E 00 00
	UTF-32	00 00 FE FF 00 00 4E 8C FF FE 00 00 8C 4E 00 00 00 00 4E 8C
00010302	UTF-32BE	00 01 03 02
	UTF-32LE	02 03 01 00
	UTF-32	00 00 FE FF 00 01 03 02 FF FE 00 00 02 03 01 00 00 01 03 02

The terms *UTF-8*, *UTF-16*, and *UTF-32*, when used unqualified, are ambiguous between their sense as Unicode encoding forms or Unicode encoding schemes. For UTF-8, this ambiguity is usually innocuous, because the UTF-8 encoding scheme is trivially derived from the byte sequences defined for the UTF-8 encoding form. However, for UTF-16 and UTF-32, the ambiguity is more problematical. As encoding forms, UTF-16 and UTF-32 refer to code units in memory; there is no associated byte orientation, and a BOM is never used. As encoding schemes, UTF-16 and UTF-32 refer to serialized bytes, as for streaming data or in files; they may have either byte orientation, and a BOM may be present.

When the usage of the short terms "UTF-16" or "UTF-32" might be misinterpreted, and where a distinction between their use as referring to Unicode encoding forms or to Unicode encoding schemes is important, the full terms, as defined in this chapter of the Unicode Standard, should be used. For example, use *UTF-16 encoding form* or *UTF-16 encoding scheme*. These terms may also be abbreviated to *UTF-16 CEF* or *UTF-16 CES*, respectively.

When converting between different encoding schemes, extreme care must be taken in handling any initial byte order marks. For example, if one converted a UTF-16 byte serialization with an initial byte order mark to a UTF-8 byte serialization, thereby converting the byte order mark to <EF BB BF> in the UTF-8 form, the <EF BB BF>

would now be ambiguous as to its status as a byte order mark (from its source) or as an initial *zero width no-break space*. If the UTF-8 byte serialization were then converted to UTF-16BE and the initial <EF BB BF> were converted to <FE FF>, the interpretation of the U+FEFF character would have been modified by the conversion. This would be nonconformant behavior according to conformance clause C7, because the change between byte serializations would have resulted in modification of the interpretation of the text. This is one reason why the use of the initial byte sequence <EF BB BF> as a signature on UTF-8 byte sequences is not recommended by the Unicode Standard.

3.11 Normalization Forms

The concepts of canonical equivalent (D70) or compatibility equivalent (D67) characters in the Unicode Standard make it necessary to have a full, formal definition of equivalence for Unicode strings. String equivalence is determined by a process called *normalization*, whereby strings are converted into forms which are compared directly for identity.

This section provides the formal definitions of the four Unicode Normalization Forms. It defines the Canonical Ordering Algorithm and the Canonical Composition Algorithm which are used to convert Unicode strings to one of the Unicode Normalization Forms for comparison. It also formally defines Unicode Combining Classes—values assigned to all Unicode characters and used by the Canonical Ordering Algorithm.

Note: In versions of the Unicode Standard up to Version 5.1.0, the Unicode Normalization Forms and the Canonical Composition Algorithm were defined in Unicode Standard Annex #15, "Unicode Normalization Forms." Those definitions have now been consolidated in this chapter, for clarity of exposition of the normative definitions and algorithms involved in Unicode normalization. However, because implementation of Unicode normalization is quite complex, implementers are still advised to fully consult Unicode Standard Annex #15, "Unicode Normalization Forms," which contains more detailed explanations, examples, and implementation strategies.

Unicode normalization should be carefully distinguished from Unicode collation. Both processes involve comparison of Unicode strings. However, the point of Unicode normalization is to make a determination of canonical (or compatibility) equivalence or non-equivalence of strings—it does not provide any rank-ordering information about those strings. Unicode collation, on the other hand, is designed to provide orderable weights or "keys" for strings; those keys can then be used to sort strings into ordered lists. Unicode normalization is not tailorable; normalization equivalence relationships between strings are exact and unchangeable. Unicode collation, on the other hand, is designed to be tailorable to allow many kinds of localized and other specialized orderings of strings. For more information, see Unicode Technical Standard #10, "Unicode Collation Algorithm."

D102 [Moved to *Section 3.6, Combination* and renumbered as D61a.]

D103 [Moved to *Section 3.6, Combination* and renumbered as D61b.]

3.11.1 Normalization Stability

A very important attribute of the Unicode Normalization Forms is that they must remain stable between versions of the Unicode Standard. A Unicode string normalized to a particular Unicode Normalization Form in one version of the standard is guaranteed to remain in that Normalization Form for implementations of future versions of the standard. In order to ensure this stability, there are strong constraints on changes of any character properties that are involved in the specification of normalization—in particular, the combining class and the decomposition of characters. The details of those constraints are spelled out in the Normalization Stability Policy. See the subsection "Policies" in *Appendix B.3, Other Unicode Online Resources*. The requirement for stability of normalization also constrains what kinds of characters can be encoded in future versions of the standard. For an extended discussion of this topic, see *Section 3, Versioning and Stability*, in Unicode Standard Annex #15, "Unicode Normalization Forms."

3.11.2 Combining Classes

Each character in the Unicode Standard has a combining class associated with it. The combining class is a numerical value used by the Canonical Ordering Algorithm to determine which sequences of combining marks are to be considered canonically equivalent and which are not. Canonical equivalence is the criterion used to determine whether two character sequences are considered identical for interpretation.

D104 Combining class: A numeric value in the range 0..254 given to each Unicode code point, formally defined as the property Canonical_Combining_Class.

- The combining class for each encoded character in the standard is specified in the file UnicodeData.txt in the Unicode Character Database. Any code point not listed in that data file defaults to Canonical_Combining_Class=0 (or ccc=0 for short).

- An extracted listing of combining classes, sorted by numeric value, is provided in the file DerivedCombiningClass.txt in the Unicode Character Database.

- Only combining marks have a combining class other than zero. Almost all combining marks with a class other than zero are also nonspacing marks, with a few exceptions. Also, not all nonspacing marks have a non-zero combining class. Thus, while the correlation between ^\p{ccc=0} and \p{gc=Mn} is close, it is not exact, and implementations should not depend on the two concepts being identical.

D105 *Fixed position class:* A subset of the range of numeric values for combining classes—specifically, any value in the range 10..199.

- Fixed position classes are assigned to a small number of Hebrew, Arabic, Syriac, Telugu, Thai, Lao, and Tibetan combining marks whose positions were conceived of as occurring in a fixed position with respect to their grapheme base, regardless of any other combining mark that might also apply to the grapheme base.

- Not all Arabic vowel points or Indic matras are given fixed position classes. The existence of fixed position classes in the standard is an historical artifact of an earlier stage in its development, prior to the formal standardization of the Unicode Normalization Forms.

D106 *Typographic interaction:* Graphical application of one nonspacing mark in a position relative to a grapheme base that is already occupied by another nonspacing mark, so that some rendering adjustment must be done (such as default stacking or side-by-side placement) to avoid illegible overprinting or crashing of glyphs.

The assignment of combining class values for Unicode characters was originally done with the goal in mind of defining distinct numeric values for each group of nonspacing marks that would typographically interact. Thus all generic nonspacing marks placed above the base character are given the same value, ccc=230, while all generic nonspacing marks placed below are given the value ccc=220. Nonspacing marks that tend to sit on one "shoulder" or another of a grapheme base, or that may actually be attached to the grapheme base itself when applied, have their own combining classes.

The design of canonical ordering generally assures that:

- When two combining characters C1 and C2 *do* typographically interact, the sequence C1+ C2 is *not* canonically equivalent to C2+ C1.

- When two combining characters C1 and C2 do *not* typographically interact, the sequence C1+ C2 *is* canonically equivalent to C2+ C1.

This is roughly correct for the normal cases of detached, generic nonspacing marks placed above and below base letters. However, the ramifications of complex rendering for many scripts ensure that there are always some edge cases involving typographic interaction between combining marks of distinct combining classes. This has turned out to be particularly true for some of the fixed position classes for Hebrew and Arabic, for which a distinct combining class is no guarantee that there will be no typographic interaction for rendering.

Because of these considerations, particular combining class values should be taken only as a guideline regarding issues of typographic interaction of combining marks.

The only *normative* use of combining class values is as input to the Canonical Ordering Algorithm, where they are used to normatively distinguish between sequences of combining marks that are canonically equivalent and those that are not.

3.11.3 Specification of Unicode Normalization Forms

The specification of Unicode Normalization Forms applies to all Unicode coded character sequences (D12). For clarity of exposition in the definitions and rules specified here, the terms "character" and "character sequence" are used, but coded character sequences refer also to sequences containing noncharacters or reserved code points. Unicode Normalization Forms are specified for *all* Unicode code points, and not just for ordinary, assigned graphic characters.

3.11.4 Starters

D107 Starter: Any code point (assigned or not) with combining class of zero (ccc = 0).

• Note that ccc = 0 is the default value for the Canonical_Combining_Class property, so that all reserved code points are Starters by definition. Noncharacters are also Starters by definition. All control characters, format characters, and private-use characters are also Starters.

• Private agreements cannot override the value of the Canonical_Combining_Class property for private-use characters.

Among the graphic characters, all those with General_Category values other than gc = M are Starters. Some combining marks have ccc = 0 and thus are also Starters. Combining marks with ccc other than 0 are not Starters. *Table 3-14* summarizes the relationship between types of combining marks and their status as Starters.

Table 3-14. Combining Marks and Starter Status

Description	gc	ccc	Starter
Nonspacing	Mn	0	Yes
		> 0	No
Spacing	Mc	0	Yes
		> 0	No
Enclosing	Me	0	Yes

The term *Starter* refers, in concept, to the starting character of a combining character sequence (D56), because all combining character sequences except defective combining character sequences (D57) commence with a ccc = 0 character—in other words, they start with a Starter. However, because the specification of Unicode Normalization Forms must apply to all possible coded character sequences, and not

just to typical combining character sequences, the behavior of a code point for Unicode Normalization Forms is specified entirely in terms of its status as a Starter or a non-starter, together with its Decomposition_Mapping value.

3.11.5 Canonical Ordering Algorithm

D108 Reorderable pair: Two adjacent characters A and B in a coded character sequence <A, B> are a Reorderable Pair if and only if ccc(A) > ccc(B) > 0.

D109 Canonical Ordering Algorithm: In a decomposed character sequence D, exchange the positions of the characters in each Reorderable Pair until the sequence contains no more Reorderable Pairs.

- In effect, the Canonical Ordering Algorithm is a local bubble sort that guarantees that a Canonical Decomposition or a Compatibility Decomposition will contain no subsequences in which a combining mark is *followed* directly by another combining mark that has a lower, non-zero combining class.

- Canonical ordering is defined in terms of application of the Canonical Ordering Algorithm to an *entire* decomposed sequence. For example, canonical decomposition of the sequence <U+1E0B LATIN SMALL LETTER D WITH DOT ABOVE, U+0323 COMBINING DOT BELOW> would result in the sequence <U+0064 LATIN SMALL LETTER D, U+0307 COMBINING DOT ABOVE, U+0323 COMBINING DOT BELOW>, a sequence which is not yet in canonical order. Most decompositions for Unicode strings are already in canonical order.

Table 3-15 gives some examples of sequences of characters, showing which of them constitute a Reorderable Pair and the reasons for that determination. Except for the base character "a", the other characters in the example table are combining marks; character names are abbreviated in the Sequence column to make the examples clearer.

Table 3-15. Reorderable Pairs

Sequence	Combining Classes	Reorderable?	Reason
<a, acute>	0, 230	No	ccc(A) = 0
<acute, a>	230, 0	No	ccc(B) = 0
<diaeresis, acute>	230, 230	No	ccc(A) = ccc(B)
<cedilla, acute>	202, 230	No	ccc(A) < ccc(B)
<acute, cedilla>	230, 202	Yes	ccc(A) > ccc(B)

3.11.6 Canonical Composition Algorithm

D110   *Singleton decomposition:* A canonical decomposition mapping from a character to a different single character.

- The default value for the Decomposition_Mapping property for a code point (including any private-use character, any noncharacter, and any unassigned code point) is the code point itself. This default value does not count as a singleton decomposition, because it does not map a character to a *different* character. Private agreements cannot override the decomposition mapping for private-use characters

- Example: U+2126 OHM SIGN has a singleton decomposition to U+03A9 GREEK CAPITAL LETTER OMEGA.

- A character with a singleton decomposition is often referred to simply as a *singleton* for short.

D110a   *Expanding canonical decomposition*: A canonical decomposition mapping from a character to a sequence of more than one character.

D110b   *Starter decomposition*: An expanding canonical decomposition for which both the character being mapped and the first character of the resulting sequence are Starters.

- Definitions D110a and D110b are introduced to simplify the following definition of non-starter decomposition and make it more precise.

D111   *Non-starter decomposition:* An expanding canonical decomposition which is not a starter decomposition.

- Example: U+0344 COMBINING GREEK DIALYTIKA TONOS has an expanding canonical decomposition to the sequence <U+0308 COMBINING DIAERESIS, U+0301 COMBINING ACUTE ACCENT>. U+0344 is a non-starter, and the first character in its decomposition is a non-starter. Therefore, on two counts, U+0344 has a non-starter decomposition.

- Example: U+0F73 TIBETAN VOWEL SIGN II has an expanding canonical decomposition to the sequence <U+0F71 TIBETAN VOWEL SIGN AA, U+0F72 TIBETAN VOWEL SIGN I>. The first character in that sequence is a non-starter. Therefore U+0F73 has a non-starter decomposition, even though U+0F73 is a Starter.

- As of the current version of the standard, there are no instances of the third possible situation: a non-starter character with an expanding canonical decomposition to a sequence whose first character is a Starter.

D112 *Composition exclusion:* A Canonical Decomposable Character (D69) which has the property value Composition_Exclusion = True.

- The list of Composition Exclusions is provided in CompositionExclusions.txt in the Unicode Character Database.

D113 *Full composition exclusion:* A Canonical Decomposable Character which has the property value Full_Composition_Exclusion = True.

- Full composition exclusions consist of the entire list of composition exclusions plus all characters with singleton decompositions or with non-starter decompositions.
- For convenience in implementation of Unicode normalization, the derived property Full_Composition_Exclusion is computed, and all characters with the property value Full_Composition_Exclusion = True are listed in DerivedNormalizationProps.txt in the Unicode Character Database.

D114 *Primary composite:* A Canonical Decomposable Character (D69) which is not a Full Composition Exclusion.

- For any given version of the Unicode Standard, the list of primary composites can be computed by extracting all canonical decomposable characters from UnicodeData.txt in the Unicode Character Database, adding the list of precomposed Hangul syllables (D132), and subtracting the list of Full Composition Exclusions.

D115 *Blocked:* Let A and C be two characters in a coded character sequence <A, ... C>. C is blocked from A if and only if ccc(A) = 0 and there exists some character B between A and C in the coded character sequence, i.e., <A, ... B, ... C>, and either ccc(B) = 0 or ccc(B) >= ccc(C).

- Because the Canonical Composition Algorithm operates on a string which is already in canonical order, testing whether a character is blocked requires looking only at the immediately preceding character in the string.

D116 *Non-blocked pair:* A pair of characters <A, ... C> in a coded character sequence, in which C is not blocked from A.

- It is important for proper implementation of the Canonical Composition Algorithm to be aware that a Non-blocked Pair need not be contiguous.

D117 *Canonical Composition Algorithm:* Starting from the second character in the coded character sequence (of a Canonical Decomposition or Compatibility Decomposition) and proceeding sequentially to the final character, perform the following steps:

R1 Seek back (left) in the coded character sequence from the character C to find the last Starter L preceding C in the character sequence.

R2 If there is such an L, and C is not blocked from L, and there exists a Primary Composite P which is canonically equivalent to the sequence <L, C>, then replace L by P in the sequence and delete C from the sequence.

- When the algorithm completes, all Non-blocked Pairs canonically equivalent to a Primary Composite will have been systematically replaced by those Primary Composites.

- The replacement of the Starter L in *R2* requires continuing to check the succeeding characters until the character at that position is no longer part of any Non-blocked Pair that can be replaced by a Primary Composite. For example, consider the following hypothetical coded character sequence: <U+007A *z*, U+0335 *short stroke overlay*, U+0327 *cedilla*, U+0324 *diaeresis below*, U+0301 *acute*>. None of the first three combining marks forms a Primary Composite with the letter *z*. However, the fourth combining mark in the sequence, *acute*, does form a Primary Composite with *z*, and it is not Blocked from the *z*. Therefore, *R2* mandates the replacement of the sequence <U+007A *z*, ... U+0301 *acute*> with <U+017A *z-acute*, ...>, even though there are three other combining marks intervening in the sequence.

- The character C in *R1* is not necessarily a non-starter. It is necessary to check *all* characters in the sequence, because there are sequences <L, C> where *both* L and C are Starters, yet there is a Primary Composite P which is canonically equivalent to that sequence. For example, Indic two-part vowels often have canonical decompositions into sequences of two spacing vowel signs, each of which has Canonical_Combining_Class = 0 and which is thus a Starter by definition. Nevertheless, such a decomposed sequence has an equivalent Primary Composite.

3.11.7 Definition of Normalization Forms

The Unicode Standard specifies four normalization forms. Informally, two of these forms are defined by maximal *decomposition* of equivalent sequences, and two of these forms are defined by maximal *composition* of equivalent sequences. Each is then differentiated based on whether it employs a Canonical Decomposition or a Compatibility Decomposition.

D118 *Normalization Form D (NFD):* The Canonical Decomposition of a coded character sequence.

D119 *Normalization Form KD (NFKD):* The Compatibility Decomposition of a coded character sequence.

D120 Normalization Form C (NFC): The Canonical Composition of the Canonical Decomposition of a coded character sequence.

D121 Normalization Form KC (NFKC): The Canonical Composition of the Compatibility Decomposition of a coded character sequence.

Logically, to get the NFD or NFKD (maximally decomposed) normalization form for a Unicode string, one first computes the full decomposition of that string and then applies the Canonical Ordering Algorithm to it.

Logically, to get the NFC or NFKC (maximally composed) normalization form for a Unicode string, one first computes the NFD or NFKD normalization form for that string, and then applies the Canonical Composition Algorithm to it.

3.12 Conjoining Jamo Behavior

The Unicode Standard contains both a large set of precomposed modern Hangul syllables and a set of conjoining Hangul jamo, which can be used to encode archaic Korean syllable blocks as well as modern Korean syllable blocks. This section describes how to

- Determine the canonical decomposition of precomposed Hangul syllables.
- Compose jamo characters into precomposed Hangul syllables.
- Algorithmically determine the names of precomposed Hangul syllables.

For more information, see the "Hangul Syllables" and "Hangul Jamo" subsections in _Section 18.6, Hangul_. Hangul syllables are a special case of grapheme clusters. For the algorithm to determine syllable boundaries in a sequence of conjoining jamo characters, see Section 8, "Hangul Syllable Boundary Determination" in Unicode Standard Annex #29, "Unicode Text Segmentation."

3.12.1 Definitions

The following definitions use the Hangul_Syllable_Type property, which is defined in the UCD file HangulSyllableType.txt.

D122　*Leading consonant:* A character with the Hangul_Syllable_Type property value Leading_Jamo. Abbreviated as L.

- When not occurring in clusters, the term _leading consonant_ is equivalent to _syllable-initial character_.

D123　*Choseong:* A sequence of one or more leading consonants.

- In Modern Korean, a _choseong_ consists of a single jamo. In Old Korean, a sequence of more than one leading consonant may occur.
- Equivalent to _syllable-initial cluster_.

D124　*Choseong filler:* U+115F HANGUL CHOSEONG FILLER. Abbreviated as L_f.

- A _choseong filler_ stands in for a missing choseong to make a well-formed Korean syllable.

D125　*Vowel:* A character with the Hangul_Syllable_Type property value Vowel_Jamo. Abbreviated as V.

- When not occurring in clusters, the term _vowel_ is equivalent to _syllable-peak character_.

D126 Jungseong: A sequence of one or more vowels.

- In Modern Korean, a *jungseong* consists of a single jamo. In Old Korean, a sequence of more than one vowel may occur.

- Equivalent to *syllable-peak cluster*.

D127 Jungseong filler: U+1160 ʜᴀɴɢᴜʟ ᴊᴜɴɢsᴇᴏɴɢ ꜰɪʟʟᴇʀ. Abbreviated as V_f.

- A *jungseong filler* stands in for a missing jungseong to make a well-formed Korean syllable.

D128 Trailing consonant: A character with the Hangul_Syllable_Type property value Trailing_Jamo. Abbreviated as T.

- When not occurring in clusters, the term *trailing consonant* is equivalent to *syllable-final character*.

D129 Jongseong: A sequence of one or more trailing consonants.

- In Modern Korean, a *jongseong* consists of a single jamo. In Old Korean, a sequence of more than one trailing consonant may occur.

- Equivalent to *syllable-final cluster*.

D130 LV_Syllable: A character with Hangul_Syllable_Type property value LV_Syllable. Abbreviated as LV.

- An LV_Syllable has a canonical decomposition to a sequence of the form <L, V>.

D131 LVT_Syllable: A character with Hangul_Syllable_Type property value LVT_Syllable. Abbreviated as LVT.

- An LVT_Syllable has a canonical decomposition to a sequence of the form <LV, T>.

D132 Precomposed Hangul syllable: A character that is either an LV_Syllable or an LVT_Syllable.

D133 Syllable block: A sequence of Korean characters that should be grouped into a single square cell for display.

- This is different from a precomposed Hangul syllable and is meant to include sequences needed for the representation of Old Korean syllables.

- A syllable block may contain a precomposed Hangul syllable *plus* other characters.

D134 *Standard Korean syllable block:* A sequence of one or more L followed by a sequence of one or more V and a sequence of zero or more T, or any other sequence that is canonically equivalent.

- All precomposed Hangul syllables, which have the form LV or LVT, are standard Korean syllable blocks.

- Alternatively, a standard Korean syllable block may be expressed as a sequence of a choseong and a jungseong, optionally followed by a jongseong.

- A choseong filler may substitute for a missing leading consonant, and a jungseong filler may substitute for a missing vowel.

- This definition is used in Unicode Standard Annex #29, "Unicode Text Segmentation," as part of the algorithm for determining syllable boundaries in a sequence of conjoining jamo characters.

3.12.2 Hangul Syllable Decomposition

The following algorithm specifies how to take a precomposed Hangul syllable *s* and arithmetically derive its full canonical decomposition *d*. This normative mapping for precomposed Hangul syllables is referenced by D68, Canonical decomposition, in *Section 3.7, Decomposition*.

This algorithm, as well as the other Hangul-related algorithms defined in the following text, is first specified in pseudo-code. Then each is exemplified, showing its application to a particular Hangul character or sequence. The Hangul characters used in those examples are shown in *Table 3-16*. Finally, each algorithm is then further exemplified with an implementation as a Java method at the end of this section.

Table 3-16. Hangul Characters Used in Examples

Code Point	Glyph	Character Name	Jamo Short Name
U+D4DB	퓛	HANGUL SYLLABLE PWILH	
U+1111	ㅍ	HANGUL CHOSEONG PHIEUPH	P
U+1171	ㅟ	HANGUL JUNGSEONG WI	WI
U+11B6	ㅀ	HANGUL JONGSEONG RIEUL-HIEUH	LH

Common Constants. Define the following constants:

$$SBase = AC00_{16}$$
$$LBase = 1100_{16}$$
$$VBase = 1161_{16}$$
$$TBase = 11A7_{16}$$
$$LCount = 19$$

```
VCount = 21
TCount = 28
NCount = 588 (VCount * TCount)
SCount = 11172 (LCount * NCount)
```

TBase is set to one less than the beginning of the range of trailing consonants, which starts at U+11A8. TCount is set to one more than the number of trailing consonants relevant to the decomposition algorithm: $(11C2_{16} - 11A8_{16} + 1) + 1$. NCount is thus the number of precomposed Hangul syllables starting with the same leading consonant, counting both the LV_Syllables and the LVT_Syllables for each possible trailing consonant. SCount is the total number of precomposed Hangul syllables.

Syllable Index. First compute the index of the precomposed Hangul syllable *s*:

```
SIndex = s – SBase
```

Arithmetic Decomposition Mapping. If the precomposed Hangul syllable *s* with the index SIndex (defined above) has the Hangul_Syllable_Type value LV, then it has a canonical decomposition mapping into a sequence of an L jamo and a V jamo, <LPart, VPart>:

```
LIndex = SIndex div NCount
VIndex = (SIndex mod NCount) div TCount
LPart = LBase + LIndex
VPart = VBase + VIndex
```

If the precomposed Hangul syllable *s* with the index SIndex (defined above) has the Hangul_Syllable_Type value LVT, then it has a canonical decomposition mapping into a sequence of an LV_Syllable and a T jamo, <LVPart, TPart>:

```
LVIndex = (SIndex div TCount) * TCount
TIndex = SIndex mod TCount
LVPart = SBase + LVIndex
TPart = TBase + TIndex
```

In this specification, the "div" operator refers to integer division (rounded down). The "mod" operator refers to the modulo operation, equivalent to the integer remainder for positive numbers.

The canonical decomposition mappings calculated this way are equivalent to the values of the Unicode character property Decomposition_Mapping (dm), for each precomposed Hangul syllable.

Full Canonical Decomposition. The full canonical decomposition for a Unicode character is defined as the recursive application of canonical decomposition mappings. The canonical decomposition mapping of an LVT_Syllable contains an LVPart which itself is a precomposed Hangul syllable and thus must be further

decomposed. However, it is simplest to unwind the recursion and directly calculate the resulting <LPart, VPart, TPart> sequence instead. For full canonical decomposition of a precomposed Hangul syllable, compute the indices and components as follows:

```
LIndex = SIndex div NCount
VIndex = (SIndex mod NCount) div TCount
TIndex = SIndex mod TCount
LPart = LBase + LIndex
VPart = VBase + VIndex
TPart = TBase + TIndex if TIndex > 0
```

If TIndex $= 0$, then there is no trailing consonant, so map the precomposed Hangul syllable *s* to its full decomposition *d* = <LPart, VPart>. Otherwise, there is a trailing consonant, so map *s* to its full decomposition *d* = <LPart, VPart, TPart>.

Example. For the precomposed Hangul syllable U+D4DB, compute the indices and components:

```
SIndex = 10459
LIndex = 17
VIndex = 16
TIndex = 15
LPart = LBase + 17 = 1111₁₆
VPart = VBase + 16 = 1171₁₆
TPart = TBase + 15 = 11B6₁₆
```

Then map the precomposed syllable to the calculated sequence of components, which constitute its full canonical decomposition:

```
U+D4DB → <U+1111, U+1171, U+11B6>
```

Note that the canonical decomposition mapping for U+D4DB would be <U+D4CC, U+11B6>, but in computing the full canonical decomposition, that sequence would only be an intermediate step.

3.12.3 Hangul Syllable Composition

The following algorithm specifies how to take a canonically decomposed sequence of Hangul jamo characters *d* and arithmetically derive its mapping to an equivalent precomposed Hangul syllable *s*. This normative mapping can be used to calculate the Primary Composite for a sequence of Hangul jamo characters, as specified in D117, Canonical Composition Algorithm, in *Section 3.11, Normalization Forms*. Strictly speaking, this algorithm is simply the inverse of the full canonical decomposition mappings specified by the Hangul Syllable Decomposition Algorithm. However, it is

useful to have a summary specification of that inverse mapping as a separate algorithm, for convenience in implementation.

Note that the presence of any non-jamo starter or any combining character between two of the jamos in the sequence *d* would constitute a blocking context, and would prevent canonical composition. See D115, Blocked, in *Section 3.11, Normalization Forms*.

Arithmetic Primary Composite Mapping. Given a Hangul jamo sequence <LPart, VPart>, where the LPart is in the range U+1100..U+1112, and where the VPart is in the range U+1161..U+1175, compute the indices and syllable mapping:

```
LIndex = LPart - LBase
VIndex = VPart - VBase
LVIndex = LIndex * NCount + VIndex * TCount
s = SBase + LVIndex
```

Given a Hangul jamo sequence <LPart, VPart, TPart>, where the LPart is in the range U+1100..U+1112, where the VPart is in the range U+1161..U+1175, and where the TPart is in the range U+11A8..U+11C2, compute the indices and syllable mapping:

```
LIndex = LPart - LBase
VIndex = VPart - VBase
TIndex = TPart - TBase
LVIndex = LIndex * NCount + VIndex * TCount
s = SBase + LVIndex + TIndex
```

The mappings just specified deal with canonically decomposed sequences of Hangul jamo characters. However, for completeness, the following mapping is also defined to deal with cases in which Hangul data is not canonically decomposed. Given a sequence <LVPart, TPart>, where the LVPart is a precomposed Hangul syllable of Hangul_Syllable_Type LV, and where the TPart is in the range U+11A8..U+11C2, compute the index and syllable mapping:

```
TIndex = TPart - TBase
s = LVPart + TIndex
```

Example. For the canonically decomposed Hangul jamo sequence <U+1111, U+1171, U+11B6>, compute the indices and syllable mapping:

```
LIndex = 17
VIndex = 16
TIndex = 15
LVIndex = 17 * 588 + 16 * 28 = 9996 + 448 = 10444
s = AC00₁₆ + 10444 + 15 = D4DB₁₆
```

Then map the Hangul jamo sequence to this precomposed Hangul syllable as its Primary Composite:

```
<U+1111, U+1171, U+11B6> → U+D4DB
```

3.12.4 Hangul Syllable Name Generation

The Unicode character names for precomposed Hangul syllables are derived algorithmically from the Jamo_Short_Name property values for each of the Hangul jamo characters in the full canonical decomposition of that syllable. That derivation is specified here.

Full Canonical Decomposition. First construct the full canonical decomposition d for the precomposed Hangul syllable s, as specified by the Hangul Syllable Decomposition Algorithm:

```
s → d = <LPart, VPart, (TPart)>
```

Jamo Short Name Mapping. For each part of the full canonical decomposition d, look up the Jamo_Short_Name property value, as specified in Jamo.txt in the Unicode Character Database. If there is no TPart in the full canonical decomposition, then the third value is set to be a null string:

```
JSNL = Jamo_Short_Name(LPart)
JSNV = Jamo_Short_Name(VPart)
JSNT = Jamo_Short_Name(TPart) if TPart exists, else ""
```

Name Concatenation. The Unicode character name for s is then constructed by starting with the constant string "HANGUL SYLLABLE" and then concatenating each of the three Jamo short name values, in order:

```
Name = "HANGUL SYLLABLE " + JSN_L + JSN_V + JSN_T
```

Example. For the precomposed Hangul syllable U+D4DB, construct the full canonical decomposition:

```
U+D4DB → <U+1111, U+1171, U+11B6>
```

Look up the Jamo_Short_Name values for each of the Hangul jamo in the canonical decomposition:

```
JSN_L = Jamo_Short_Name(U+1111) = "P"
JSN_V = Jamo_Short_Name(U+1171) = "WI"
JSN_T = Jamo_Short_Name(U+11B6) = "LH"
```

Concatenate the pieces:

```
Name = "HANGUL SYLLABLE " + "P" + "WI" + "LH"
       = "HANGUL SYLLABLE PWILH"
```

3.12.5 Sample Code for Hangul Algorithms

This section provides sample Java code illustrating the three Hangul-related algorithms.

Common Constants. This code snippet defines the common constants used in the methods that follow.

```
static final int
    SBase = 0xAC00,
    LBase = 0x1100, VBase = 0x1161, TBase = 0x11A7,
    LCount = 19, VCount = 21, TCount = 28,
    NCount = VCount * TCount,   // 588
    SCount = LCount * NCount;   // 11172
```

Hangul Decomposition. The Hangul Decomposition Algorithm as specified above directly decomposes precomposed Hangul syllable characters into a sequence of either two or three Hangul jamo characters. The sample method here does precisely that:

```
public static String decomposeHangul(char s) {
    int SIndex = s - SBase;
    if (SIndex < 0 || SIndex >= SCount) {
      return String.valueOf(s);
    }
    StringBuffer result = new StringBuffer();
    int L = LBase + SIndex / NCount;
    int V = VBase + (SIndex % NCount) / TCount;
    int T = TBase + SIndex % TCount;
    result.append((char)L);
    result.append((char)V);
    if (T != TBase) result.append((char)T);
    return result.toString();
  }
```

The Hangul Decomposition Algorithm could also be expressed equivalently as a recursion of binary decompositions, as is the case for other non-Hangul characters. All LVT syllables would decompose into an LV syllable plus a T jamo. The LV syllables themselves would in turn decompose into an L jamo plus a V jamo. This approach can be used to produce somewhat more compact code than what is illustrated in this sample method.

Hangul Composition. An important feature of Hangul composition is that whenever the source string is not in Normalization Form D or Normalization Form KD, one must not detect only character sequences of the form <L, V> and <L, V, T>. It is

also necessary to catch the sequences of the form <LV, T>. To guarantee uniqueness, such sequences must also be composed. This extra processing is illustrated in step 2 of the sample method defined here.

```
public static String composeHangul(String source) {
    int len = source.length();
    if (len == 0) return "";
    StringBuffer result = new StringBuffer();
    char last = source.charAt(0);        // copy first char
    result.append(last);

    for (int i = 1; i < len; ++i) {
      char ch = source.charAt(i);

      // 1. check to see if two current characters are L and V
      int LIndex = last - LBase;
      if (0 <= LIndex && LIndex < LCount) {
        int VIndex = ch - VBase;
        if (0 <= VIndex && VIndex < VCount) {

          // make syllable of form LV

          last = (char)(SBase + (LIndex * VCount + VIndex)
          * TCount);

          result.setCharAt(result.length()-1, last); // reset last
          continue; // discard ch
          }
      }

      // 2. check to see if two current characters are LV and T
      int SIndex = last - SBase;
      if (0 <= SIndex && SIndex < SCount
        && (SIndex % TCount) == 0) {
        int TIndex = ch - TBase;
        if (0 < TIndex && TIndex < TCount) {

          // make syllable of form LVT

          last += TIndex;
          result.setCharAt(result.length()-1, last); // reset last
          continue; // discard ch
        }
      }
      // if neither case was true, just add the character
      last = ch;
      result.append(ch);
    }
```

```
        return result.toString();
    }
```

Hangul Character Name Generation. Hangul decomposition is also used when generating the names for precomposed Hangul syllables. This is apparent in the following sample method for constructing a Hangul syllable name. The content of the three tables used in this method can be derived from the data file Jamo.txt in the Unicode Character Database.

```
public static String getHangulName(char s) {
    int SIndex = s – SBase;
    if (0 > SIndex || SIndex >= SCount) {
      throw new IllegalArgumentException("Not a Hangul Syllable: "
        + s);
    }
    int LIndex = SIndex / NCount;
    int VIndex = (SIndex % NCount) / TCount;
    int TIndex = SIndex % TCount;
    return "HANGUL SYLLABLE " + JAMO_L_TABLE[LIndex]
      + JAMO_V_TABLE[VIndex] + JAMO_T_TABLE[TIndex];
}

  static private String[] JAMO_L_TABLE = {
    "G", "GG", "N", "D", "DD", "R", "M", "B", "BB",
    "S", "SS", "", "J", "JJ", "C", "K", "T", "P", "H"
  };

  static private String[] JAMO_V_TABLE = {
    "A", "AE", "YA", "YAE", "EO", "E", "YEO", "YE", "O",
    "WA", "WAE", "OE", "YO", "U", "WEO", "WE", "WI",
    "YU", "EU", "YI", "I"
  };

  static private String[] JAMO_T_TABLE = {
    "", "G", "GG", "GS", "N", "NJ", "NH", "D", "L", "LG", "LM",
    "LB", "LS", "LT", "LP", "LH", "M", "B", "BS",
    "S", "SS", "NG", "J", "C", "K", "T", "P", "H"
  };
```

Additional Transformations for Hangul Jamo. Additional transformations can be performed on sequences of Hangul jamo for various purposes. For example, to regularize sequences of Hangul jamo into standard Korean syllable blocks, the *choseong* or *jungseong* fillers can be inserted, as described in Unicode Standard Annex #29, "Unicode Text Segmentation."

For keyboard input, additional compositions may be performed. For example, a sequence of trailing consonants $k_f + s_f$ may be combined into a single, complex jamo ks_f. In addition, some Hangul input methods do not require a distinction on input

between initial and final consonants, and may instead change between them on the basis of context. For example, in the keyboard sequence $m_i + e_m + n_i + s_i + a_m$, the consonant n_i would be reinterpreted as n_f, because there is no possible syllable *nsa*. This results in the two syllables *men* and *sa*.

3.13 Default Case Algorithms

This section specifies the default algorithms for case conversion, case detection, and caseless matching. For information about the data sources for case mapping, see *Section 4.2, Case*. For a general discussion of case mapping operations, see *Section 5.18, Case Mappings*.

All of these specifications are *logical* specifications. Particular implementations can optimize the processes as long as they provide the same results.

Tailoring. The default casing operations are intended for use in the *absence* of tailoring for particular languages and environments. Where a particular environment requires tailoring of casing operations to produce correct results, use of such tailoring does not violate conformance to the standard.

Data that assist the implementation of certain tailorings are published in Special-Casing.txt in the Unicode Character Database. Most notably, these include:

- Casing rules for the Turkish *dotted capital I* and *dotless small i*.
- Casing rules for the retention of dots over *i* for Lithuanian letters with additional accents.

Examples of case tailorings which are not covered by data in SpecialCasing.txt include:

- Titlecasing of IJ at the start of words in Dutch
- Removal of accents when uppercasing letters in Greek
- Titlecasing of second or subsequent letters in words in orthographies that include caseless letters such as apostrophes
- Uppercasing of U+00DF "ß" LATIN SMALL LETTER SHARP S to U+1E9E LATIN CAPITAL LETTER SHARP S

The preferred mechanism for defining tailored casing operations is the Unicode Common Locale Data Repository (CLDR), https://cldr.unicode.org, where tailorings such as these can be specified on a per-language basis, as needed.

Tailorings of case operations may or may not be desired, depending on the nature of the implementation in question. For more about complications in case mapping, see the discussion in *Section 5.18, Case Mappings*.

3.13.1 Definitions

The full case mappings for Unicode characters are obtained by using the mappings from SpecialCasing.txt *plus* the mappings from UnicodeData.txt, excluding any of the latter mappings that would conflict. Any character that does not have a mapping in these files is considered to map to itself. The full case mappings of a character C are referred to as Lowercase_Mapping(C), Titlecase_Mapping(C), and Uppercase_Mapping(C). The full case folding of a character C is referred to as Case_Folding(C).

Detection of case and case mapping requires more than just the General_Category values (Lu, Lt, Ll). The following definitions are used:

D135   A character C is defined to be *cased* if and only if C has the Lowercase or Uppercase property or has a General_Category value of Titlecase_Letter.

     • The Uppercase and Lowercase property values are specified in the data file DerivedCoreProperties.txt in the Unicode Character Database. The derived property Cased is also listed in DerivedCoreProperties.txt.

D136   A character C is defined to be *case-ignorable* if C has the value MidLetter (ML), MidNumLet (MB), or Single_Quote (SQ) for the Word_Break property or its General_Category is one of Nonspacing_Mark (Mn), Enclosing_Mark (Me), Format (Cf), Modifier_Letter (Lm), or Modifier_Symbol (Sk).

     • The Word_Break property is defined in the data file WordBreakProperty.txt in the Unicode Character Database.

     • The derived property Case_Ignorable is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

     • The Case_Ignorable property is defined for use in the context specifications of *Table 3-17*. It is a narrow-use property, and is not intended for use in other contexts. The more broadly applicable string casing function, isCased(X), is defined in D143.

D137   C*ase-ignorable sequence*: A sequence of zero or more case-ignorable characters.

D138   A character C is in a particular *casing context* for context-dependent matching if and only if it matches the corresponding specification in *Table 3-17*.

Table 3-17. Context Specification for Casing

Context	Description	Regular Expressions

Final_Sigma	C is preceded by a sequence consisting of a cased letter and then zero or more case-ignorable characters, and C is not followed by a sequence consisting of zero or more case-ignorable characters and then a cased letter.	Before C	\p{cased} (\p{Case_Ignorable})*
		After C	! ((\p{Case_Ignorable})* \p{cased})
After_Soft_Dotted	There is a Soft_Dotted character before C, with no intervening character of combining class 0 or 230 (Above).	Before C	[\p{Soft_Dotted}] ([^\p{ccc=230} \p{ccc=0}])*
More_Above	C is followed by a character of combining class 230 (Above) with no intervening character of combining class 0 or 230 (Above).	After C	[^\p{ccc=230}\p{ccc=0}]* [\p{ccc=230}]
Before_Dot	C is followed by COMBINING DOT ABOVE (U+0307). Any sequence of characters with a combining class that is neither 0 nor 230 may intervene between the current character and the *combining dot above*.	After C	([^\p{ccc=230} \p{ccc=0}])* [\u0307]
After_I	There is an uppercase I before C, and there is no intervening combining character class 230 (Above) or 0.	Before C	[I] ([^\p{ccc=230} \p{ccc=0}])*

In *Table 3-17*, a description of each context is followed by the equivalent regular expression(s) describing the context before C, the context after C, or both. The regular expressions use the syntax of Unicode Technical Standard #18, "Unicode Regular Expressions," with one addition: "!" means that the expression does not match. All of the regular expressions are case-sensitive.

The regular-expression operator * in *Table 3-17* is "possessive," consuming as many characters as possible, with no backup. This is significant in the case of Final_Sigma, because the sets of case-ignorable and cased characters are not disjoint: for example, they both contain U+0345 COMBINING GREEK YPOGEGRAMMENI. Thus, the Before condition is not satisfied if C is preceded by only U+0345, but would be satisfied by the sequence <*capital-alpha, ypogegrammeni*>. Similarly, the After condition is satisfied if C is only followed by *ypogegrammeni*, but would not satisfied by the sequence <*ypogegrammeni, capital-alpha*>.

Additional language-specific or orthography-specific contexts and casing behavior is specified in the Unicode Common Locale Data Repository (CLDR), https://cldr.unicode.org.

3.13.2 Default Case Conversion

The following rules specify the default case conversion operations for Unicode strings. These rules use the full case conversion operations, Uppercase_Mapping(C), Lowercase_Mapping(C), and Titlecase_Mapping(C), as well as the context-dependent mappings based on the casing context, as specified in *Table 3-17*.

For a string X:

> **R1 toUppercase(X): Map each character C in X to Uppercase_Mapping(C).**

> **R2 toLowercase(X): Map each character C in X to Lowercase_Mapping(C).**

> **R3 toTitlecase(X): Find the word boundaries in X according to Unicode Standard Annex #29, "Unicode Text Segmentation." For each word boundary, find the first cased character F following the word boundary. If F exists, map F to Titlecase_Mapping(F); then map all characters C between F and the following word boundary to Lowercase_Mapping(C).**

The default case conversion operations may be tailored for specific requirements. A common variant, for example, is to make use of simple case conversion, rather than full case conversion. Language- or locale-specific tailorings of these rules may also be used.

3.13.3 Default Case Folding

Case folding is related to case conversion. However, the main purpose of case folding is to contribute to caseless matching of strings, whereas the main purpose of case conversion is to put strings into a particular cased form.

Default Case Folding does not preserve normalization forms. A string in a particular Unicode normalization form may not be in that normalization form after it has been casefolded.

Default Case Folding is based on the full case conversion operations without the context-dependent mappings sensitive to the casing context. There are also some adaptations specifically to support caseless matching. Lowercase_Mapping(C) is used for most characters, but there are instances in which the folding must be based on Uppercase_Mapping(C), instead. In particular, the addition of lowercase Cherokee letters as of Version 8.0 of the Unicode Standard, together with the stability guarantees for case folding, require that Cherokee letters be case folded to their *uppercase* counterparts. *As a result, a case folded string is not necessarily lowercase.*

Any two strings which are considered to be case variants of each other under any of the full case conversions, toUppercase(X), toLowercase(X), or toTitlecase(X) will fold to the same string by the toCasefold(X) operation:

R4 toCasefold(X): Map each character C in X to Case_Folding(C).

- Case_Folding(C) uses the mappings with the status field value "C" and "F" in the data file CaseFolding.txt in the Unicode Character Database.

- *Simple case folding* is a modification of Default Case Folding which maps code points to single code points. It consists in using the Simple_Case_Folding property instead of the Case_Folding property. Simple case folding is used in some applications requiring compatibility with legacy definitions of case-insensitivity.

- The Simple_Case_Folding property uses the mappings with the status field value "C" and "S".

A modified form of Default Case Folding is designed for best behavior when doing caseless matching of strings interpreted as identifiers. This folding is based on Case_Folding(C), but also removes any characters which have the Unicode property value Default_Ignorable_Code_Point = True. It also maps characters to their NFKC equivalent sequences. Once the mapping for a string is complete, the resulting string is then normalized to NFC. That last normalization step simplifies the statement of the use of this folding for caseless matching.

R5 toNFKC_Casefold(X): Map each character C in X to NFKC_Casefold(C) and then normalize the resulting string to NFC.

- The mapping NFKC_Casefold (short alias NFKC_CF) is specified in the data file DerivedNormalizationProps.txt in the Unicode Character Database.

- The derived binary property Changes_When_NFKC_Casefolded is also listed in the data file DerivedNormalizationProps.txt in the Unicode Character Database.

The Default_Ignorable_Code_Point property values used to derive NFKC_Casefold are not guaranteed to be stable. However, the derivation of NFKC_Casefold will be changed as necessary to ensure that it remains stable for default identifiers. Thus, if the NFKC_Casefold operation is applied to a string containing only characters from XID_Continue in a given version of the Unicode Standard, the result will be the same in any future version.

For more information on the use of NFKC_Casefold and caseless matching for identifiers, see Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax." For the stability of NFKC_Casefold, in particular, see *Section 2.3, Layout and Format Control Characters*.

3.13.4 Default Case Detection

The casing status of a string can be determined by using the casing operations defined earlier. The following definitions provide a specification. They assume that X

and Y are strings. In the following, functional names beginning with "is" are binary functions which take the string X and return true when the string as a whole matches the given casing status. For example, isLowercase(X) would be true if the string X as a whole is lowercase. In contrast, the Unicode character properties such as Lowercase are properties of individual characters.

For each definition, there is also a related Unicode character property which has a name beginning with "Changes_When_". That property indicates whether each character is affected by a particular casing operation; it can be used to optimize implementations of Default Case Detection for strings.

When case conversion is applied to a string that is decomposed (or more precisely, normalized to NFD), applying the case conversion character by character does not affect the normalization status of the string. Therefore, these definitions are specified in terms of Normalization Form NFD. To make the definitions easier to read, they adopt the convention that the string Y equals toNFD(X).

D139 isLowercase(X): isLowercase(X) is true when toLowercase(Y) = Y.

- For example, isLowercase("combining mark") is true, and isLowercase("Combining mark") is false.
- The derived binary property Changes_When_Lowercased is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

D140 isUppercase(X): isUppercase(X) is true when toUppercase(Y) = Y.

- For example, isUppercase("COMBINING MARK") is true, and isUppercase("Combining mark") is false.
- The derived binary property Changes_When_Uppercased is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

D141 isTitlecase(X): isTitlecase(X) is true when toTitlecase(Y) = Y.

- For example, isTitlecase("Combining Mark") is true, and isTitlecase("Combining mark") is false.
- The derived binary property Changes_When_Titlecased is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

D142 isCasefolded(X): isCasefolded(X) is true when toCasefold(Y) = Y.

- For example, isCasefolded("heiss") is true, and isCasefolded("heiß") is false.
- The derived binary property Changes_When_Casefolded is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

Uncased characters do not affect the results of casing detection operations such as the string function isLowercase(X). Thus a space or a number added to a string does not affect the results.

The examples in *Table 3-18* show that these conditions are not mutually exclusive. "A2" is both uppercase and titlecase; "3" is uncased, so it is *simultaneously* lowercase, uppercase, and titlecase.

Table 3-18. Case Detection Examples

Case	Letter	Name	Alphanumeric	Digit
Lowercase	a	john smith	a2	3
Uppercase	A	JOHN SMITH	A2	3
Titlecase	A	John Smith	A2	3

Only when a string, such as "123", contains no cased letters will all three conditions, —isLowercase, isUppercase, and isTitlecase—evaluate as true. This combination of conditions can be used to check for the presence of cased letters, using the following definition:

D143 isCased(X): isCased(X) is true when isLowercase(X) is false, or isUppercase(X) is false, or isTitlecase(X) is false.

- Any string X for which isCased(X) is true contains at least one character that has a case mapping other than to itself.
- For example, isCased("123") is false because all the characters in "123" have case mappings to themselves, while isCased("abc") and isCased("A12") are both true.
- The derived binary property Changes_When_Casemapped is listed in the data file DerivedCoreProperties.txt in the Unicode Character Database.

To find out whether a string contains only lowercase letters, implementations need to test for (isLowercase(X) and isCased(X)).

3.13.5 Default Caseless Matching

Default caseless matching is the process of comparing two strings for case-insensitive equality. The definitions of Unicode Default Caseless Matching build on the definitions of Unicode Default Case Folding.

Default Caseless Matching uses full case folding:

D144 A string X is a caseless match for a string Y if and only if:
toCasefold(X) = toCasefold(Y)

When comparing strings for case-insensitive equality, the strings should also be normalized for most correct results. For example, the case folding of U+00C5 Å LATIN CAPITAL LETTER A WITH RING ABOVE is U+00E5 å LATIN SMALL LETTER A WITH RING ABOVE, whereas the case folding of the sequence <U+0041 "A" LATIN CAPITAL LETTER A, U+030A ̊ COMBINING RING ABOVE> is the sequence <U+0061 "a" LATIN SMALL LETTER A, U+030A ̊ COMBINING RING ABOVE>. Simply doing a binary comparison of the results of case folding both strings will not catch the fact that the resulting case-folded strings are canonical-equivalent sequences. In principle, normalization needs to be done after case folding, because case folding does not preserve the normalized form of strings in all instances. This requirement for normalization is covered in the following definition for canonical caseless matching:

D145 A string X is a canonical caseless match for a string Y if and only if:
NFD(toCasefold(NFD(X))) = NFD(toCasefold(NFD(Y)))

The invocations of canonical decomposition (NFD normalization) before case folding in D145 are to catch very infrequent edge cases. Normalization is not required before case folding, except for the character U+0345 ͅ COMBINING GREEK YPOGEGRAMMENI and any characters that have it as part of their canonical decomposition, such as U+1FC3 ῃ GREEK SMALL LETTER ETA WITH YPOGEGRAMMENI. In practice, optimized versions of canonical caseless matching can catch these special cases, thereby avoiding an extra normalization step for each comparison.

In some instances, implementers may wish to ignore compatibility differences between characters when comparing strings for case-insensitive equality. The correct way to do this makes use of the following definition for compatibility caseless matching:

D146 A string X is a compatibility caseless match for a string Y if and only if:
NFKD(toCasefold(NFKD(toCasefold(NFD(X))))) =
NFKD(toCasefold(NFKD(toCasefold(NFD(Y)))))

Compatibility caseless matching requires an extra cycle of case folding and normalization for each string compared, because the NFKD normalization of a compatibility character such as U+3392 SQUARE MHZ may result in a sequence of alphabetic characters which must again be case folded (and normalized) to be compared correctly.

Caseless matching for identifiers can be simplified and optimized by using the NFKC_Casefold mapping. That mapping incorporates internally the derived results of iterated case folding and NFKD normalization. It also maps away characters with the property value Default_Ignorable_Code_Point = True, which should not make a difference when comparing identifiers.

The following defines identifier caseless matching:

D147 A string X is an identifier caseless match for a string Y if and only if:
   toNFKC_Casefold(NFD(X)) = toNFKC_Casefold(NFD(Y))

Chapter 4

Character Properties

Disclaimer

The content of all character property tables has been verified as far as possible by the Unicode Consortium. However, in case of conflict, the most authoritative version of the information for this version of the Unicode Standard is that supplied in the Unicode Character Database on the Unicode website. *The contents of all the tables in this chapter may be superseded or augmented by information in future versions of the Unicode Standard.*

The Unicode Standard associates a rich set of semantics with characters and, in some instances, with code points. The support of character semantics is required for conformance; see *Section 3.2, Conformance Requirements*. Where character semantics can be expressed formally, they are provided as machine-readable lists of character properties in the Unicode Character Database (UCD). This chapter gives an overview of character properties, their status and attributes, followed by an overview of the UCD and more detailed notes on some important character properties. For a further discussion of character properties, see Unicode Technical Report #23, "Unicode Character Property Model."

Status and Attributes. Character properties may be normative, informative, contributory, or provisional. Normative properties are those required for conformance. Many Unicode character properties can be overridden by implementations as needed. *Section 3.2, Conformance Requirements*, specifies when such overrides must be documented. A few properties, such as Noncharacter_Code_Point, may not be overridden. See *Section 3.5, Properties*, for the formal discussion of the status and attributes of properties.

Consistency of Properties. The Unicode Standard is the product of many compromises. It has to strike a balance between uniformity of treatment for similar characters and compatibility with existing practice for characters inherited from legacy encodings. Because of this balancing act, one can expect a certain number of anomalies in character properties. For example, some pairs of characters might have been treated as canonical equivalents but are left unequivalent for compatibility with legacy differences. This situation pertains to U+00B5 µ MICRO SIGN and U+03BC μ GREEK SMALL LETTER MU, as well as to certain Korean jamo.

In addition, some characters might have had properties differing in some ways from those assigned in this standard, but those properties are left as is for compatibility with existing practice. This situation can be seen with the halfwidth voicing marks for Japanese (U+FF9E HALFWIDTH KATAKANA VOICED SOUND MARK and U+FF9F HALFWIDTH KATAKANA SEMI-VOICED SOUND MARK), which might have been better analyzed as spacing combining marks. Another examples consists of the conjoining Hangul jamo, which might have been better analyzed as an initial base character followed by formally combining medial and final characters. In the interest of efficiency and uniformity in algorithms, implementations may take advantage of such reanalyses of character properties, as long as this does not conflict with the conformance requirements with respect to normative properties. See *Section 3.5, Properties*; *Section 3.2, Conformance Requirements*; and *Section 3.3, Semantics*, for more information.

4.1 Unicode Character Database

The Unicode Character Database (UCD) consists of a set of files that define the Unicode character properties and internal mappings. For each property, the files determine the assignment of property values to each code point. The UCD also supplies recommended property aliases and property value aliases for textual parsing and display in environments such as regular expressions.

The properties include the following:

- Name
- General Category (basic partition into letters, numbers, symbols, punctuation, and so on)
- Other important general characteristics (whitespace, dash, ideographic, alphabetic, noncharacter, deprecated, and so on)
- Display-related properties (bidirectional class, shaping, mirroring, width, and so on)
- Casing (upper, lower, title, folding—both simple and full)
- Numeric values and types
- Script and Block
- Normalization properties (decompositions, decomposition type, canonical combining class, composition exclusions, and so on)
- Age (version of the standard in which the code point was first designated)
- Boundaries (grapheme cluster, word, line, and sentence)

See Unicode Standard Annex #44, "Unicode Character Database," for more details on the character properties and their values, the status of properties, their distribution across data files, and the file formats.

Unihan Database. In addition, a large number of properties specific to CJK ideographs are defined in the Unicode Character Database. These properties include source information, radical and stroke counts, phonetic values, meanings, and mappings to many East Asian standards. The values for all these properties are listed in the file Unihan.zip, also known as the *Unihan Database*. For a complete description and documentation of the properties themselves, see Unicode Standard Annex #38, "Unicode Han Database (Unihan)." (See also "Online Unihan Database" in *Appendix B.3, Other Unicode Online Resources*.)

Many properties apply to both ideographs and other characters. These are not specified in the Unihan Database, but rather in other data files associated with the UCD.

Unikemet. A number of properties specific to Egyptian hieroglyphs are defined in Unikemet.txt. These properties include source information, function, value, description, mirroring and rotation status. For a complete description and documentation of the properties, see Unicode Standard Annex #57, "Unicode Egyptian Hieroglyph Database (Unikemet)."

Stability. While the Unicode Consortium strives to minimize changes to character property data, occasionally character properties must be updated. When this situation occurs, a new version of the Unicode Character Database is created, containing updated data files. Data file changes are associated with specific, numbered versions of the standard; character properties are never silently corrected between official versions.

Each version of the Unicode Character Database, once published, is absolutely stable and will never change. Implementations or specifications that refer to a specific version of the UCD can rely upon this stability. Detailed policies on character encoding stability as they relate to properties are found on the Unicode website. See the subsection "Policies" in *Appendix B.3, Other Unicode Online Resources*. See also the discussion of versioning and stability in *Section 3.1, Versions of the Unicode Standard*.

Aliases. Character properties and their values are given formal aliases to make it easier to refer to them consistently in specifications and in implementations, such as regular expressions, which may use them. These aliases are listed exhaustively in the Unicode Character Database, in the data files PropertyAliases.txt and PropertyValueAliases.txt.

Many of the aliases have both a long form and a short form. For example, the General Category has a long alias "General_Category" and a short alias "gc". The long alias is more comprehensible and is usually used in the text of the standard when referring to a particular character property. The short alias is more appropriate for use in regular expressions and other algorithmic contexts.

In comparing aliases programmatically, loose matching is appropriate. That entails ignoring case differences and any whitespace, underscore, and hyphen characters. For example, "GeneralCategory", "general_category", and "GENERAL-CATEGORY" would all be considered equivalent property aliases. See Unicode Standard Annex #44, "Unicode Character Database," for further discussion of property and property value matching.

For each character property whose values are not purely numeric, the Unicode Character Database provides a list of value aliases. For example, one of the values of the Line_Break property is given the long alias "Open_Punctuation" and the short alias "OP".

Property aliases and property value aliases can be combined in regular expressions that pick out a particular value of a particular property. For example, "\p{lb=OP}" means the Open_Punctuation value of the Line_Break property, and "\p{gc=Lu}" means the Uppercase_Letter value of the General_Category property.

Property aliases define a namespace. No two character properties have the same alias. For each property, the set of corresponding property value aliases constitutes its own namespace. No constraint prevents property value aliases for *different* properties from having the same property value alias. Thus "B" is the short alias for the Paragraph_Separator value of the Bidi_Class property; "B" is also the short alias for the Below value of the Canonical_Combining_Class property. However, because of the namespace restrictions, any combination of a property alias plus an appropriate property value alias is guaranteed to constitute a unique string, as in "\p{bc=B}" versus "\p{ccc=B}".

For a recommended use of property and property value aliases, see Unicode Technical Standard #18, "Unicode Regular Expressions." Aliases are also used for normatively referencing properties, as described in *Section 3.1, Versions of the Unicode Standard*.

UCD in XML. Starting with Unicode Version 5.1.0, the complete Unicode Character Database is also available formatted in XML. This includes both the non-Han part of the Unicode Character Database and all of the content of the Unihan Database. For details regarding the XML schema, file names, grouping conventions, and other considerations, see Unicode Standard Annex #42, "Unicode Character Database in XML."

Online Availability. All versions of the UCD are available online on the Unicode website. See the subsections "Online Unicode Character Database" and "Online Unihan Database" in *Appendix B.3, Other Unicode Online Resources*.

4.2 Case

Case is a normative property of characters in certain alphabets whereby characters are considered to be variants of a single letter. These variants, which may differ markedly in shape and size, are called the *uppercase* letter (also known as *capital* or *majuscule*) and the *lowercase* letter (also known as *small* or *minuscule*). The uppercase letter is generally larger than the lowercase letter.

Because of the inclusion of certain composite characters for compatibility, such as U+01F1 ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ ᴅᴢ, a third case, called *titlecase*, is used where the first character of a word must be capitalized. An example of such a character is U+01F2 ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ ᴅ ᴡɪᴛʜ ꜱᴍᴀʟʟ ʟᴇᴛᴛᴇʀ ᴢ. The three case forms are UPPERCASE, Titlecase, and lowercase.

For those scripts that have case (such as Latin, Greek, Coptic, Cyrillic, Glagolitic, Armenian, archaic Georgian, Deseret, and Warang Citi), uppercase characters typically contain the word *capital* in their names. Lowercase characters typically contain the word *small*. However, this is not a reliable guide. The word *small* in the names of characters from scripts other than those just listed has nothing to do with case. There are other exceptions as well, such as small capital letters that are not formally uppercase. Some Greek characters with *capital* in their names are actually titlecase. (Note that while the archaic Georgian script contained upper- and lowercase pairs, they are not used in modern Georgian. See *Section 7.7, Georgian*.)

4.2.1 Definitions of Case and Casing

The Unicode Standard has more than one formal definition of lowercase, uppercase, and related casing processes. This is the result of the inherent complexity of case relationships and of defining case-related behavior on the basis of individual character properties. This section clarifies the distinctions involved in the formal definition of casing in the standard. The additional complications for titlecase are omitted from the discussion; titlecase distinctions apply only to a handful of compatibility characters.

The first set of values involved in the definition of case are based on the General_Category property in UnicodeData.txt. The relevant values are General_Category = Ll (Lowercase_Letter) and General_Category = Lu (Uppercase_Letter). For most ordinary letters of bicameral scripts such as Latin, Greek, and Cyrillic, these values are obvious and non-problematical. However, the General_Category property is, by design, a partition of the Unicode codespace. This means that each Unicode character can only have one General_Category value, which results in some odd edge cases for modifier letters, letterlike symbols and letterlike numbers. As a consequence, not every Unicode character that *looks like* a lowercase character necessarily ends up with General_Category = Ll, and not every

Unicode character that *looks like* an uppercase character ends up with General_Category = Lu.

The second set of definitions relevant to case consist of the derived binary properties, Lowercase and Uppercase, specified in DerivedCoreProperties.txt in the Unicode Character Database. Those derived properties augment the General_Category values by adding the additional characters that ordinary users think of as being lowercase or uppercase, based primarily on their letterforms. The additional characters are included in the derivations by means of the contributory properties, Other_Lowercase and Other_Uppercase, defined in PropList.txt. For example, Other_Lowercase adds the various modifier letters that are letterlike in shape, the circled lowercase letter symbols, and the compatibility lowercase Roman numerals. Other_Uppercase adds the circled uppercase letter symbols, and the compatibility uppercase Roman numerals.

A third set of definitions for case is fundamentally different in kind, and does not consist of character properties at all. The functions isLowercase and isUppercase are string functions returning a binary True/False value. These functions are defined in *Section 3.13, Default Case Algorithms*, and depend on case mapping relations, rather than being based on letterforms per se. Basically, isLowercase is True for a string if the result of applying the toLowercase mapping operation for a string is the same as the string itself.

Table 4-1 illustrates the various possibilities for how these definitions interact, as applied to exemplary single characters or single character strings.

Table 4-1. Relationship of Casing Definitions

Code	Character	gc	Lowercase	Uppercase	isLowercase(S)	isUppercase(S)
0068	h	Ll	True	False	True	False
0048	H	Lu	False	True	False	True
24D7	ⓗ	So	True	False	True	False
24BD	Ⓗ	So	False	True	False	True
02B0	ʰ	Lm	True	False	True	True
1D34	ᴴ	Lm	True	False	True	True
02BD	ʽ	Lm	False	False	True	True

Note that for "caseless" characters, such as U+02B0, U+1D34, and U+02BD, isLowercase and isUppercase are *both* True, because the inclusion of a caseless letter in a string is not criterial for determining the casing of the string—a caseless letter always case maps to itself.

On the other hand, all modifier letters derived from letter shapes are also notionally lowercase, whether the letterform itself is a minuscule or a majuscule in shape. Thus

U+1D34 MODIFIER LETTER CAPITAL H is actually Lowercase = True. Other modifier letters not derived from letter shapes, such as U+02BD, are neither Lowercase nor Uppercase.

The string functions isLowercase and isUppercase also apply to strings longer than one character, of course, for which the character properties General_Category, Lowercase, and Uppercase are not relevant. In *Table 4-2*, the string function isTitlecase is also illustrated, to show its applicability for the same strings.

Table 4-2. Case Function Values for Strings

Codes	String	isLowercase(S)	isUppercase(S)	isTitlecase(S)
0068 0068	hh	True	False	False
0048 0048	HH	False	True	False
0048 0068	Hh	False	False	True
0068 0048	hH	False	False	False

Programmers concerned with manipulating Unicode strings should generally be dealing with the string functions such as isLowercase (and its functional cousin, toLowercase), unless they are working directly with single character properties. Care is always advised, however, when dealing with case in the Unicode Standard, as expectations based simply on the behavior of the basic Latin alphabet (A..Z, a..z) do not generalize easily across the entire repertoire of Unicode characters, and because case for modifier letters, in particular, can result in unexpected behavior.

4.2.2 Case Mapping

The default case mapping tables defined in the Unicode Standard are normative, but may be overridden to match user or implementation requirements. The Unicode Character Database contains four files with case mapping information, as shown in *Table 4-3*. Full case mappings for Unicode characters are obtained by using the basic mappings from UnicodeData.txt and extending or overriding them where necessary with the mappings from SpecialCasing.txt. Full case mappings may depend on the context surrounding the character in the original string.

Some characters have a "best" single-character mapping in UnicodeData.txt as well as a full mapping in SpecialCasing.txt. Any character that does not have a mapping in these files is considered to map to itself. For more information on case mappings, see *Section 5.18, Case Mappings*.

Table 4-3. Sources for Case Mapping Information

File Name	Description

UnicodeData.txt	Contains the case mappings that map to a single character. These do not increase the length of strings, nor do they contain context-dependent mappings.
SpecialCasing.txt	Contains additional case mappings that map to more than one character, such as "ß" to "SS". Also contains context-dependent mappings, with flags to distinguish them from the normal mappings, as well as some locale-dependent mappings.
CaseFolding.txt	Contains data for performing default case folding and simple (code point-to-code point) case folding, as described in "Caseless Matching," in *Section 5.18, Case Mappings*, and in "Default Case Folding" in *Section 3.13, Default Case Algorithms*. Case folding is locale-independent.
PropList.txt	Contains the definition of the property Soft_Dotted, which is used in the context specification for casing. See D138 in *Section 3.13, Default Case Algorithms*.

The single-character mappings in UnicodeData.txt are insufficient for languages such as German. Therefore, only legacy implementations that cannot handle case mappings that increase string lengths should use UnicodeData.txt case mappings alone.

A set of charts that show the latest case mappings is also available on the Unicode website. See "Charts" in *Appendix B.3, Other Unicode Online Resources*.

4.3 Combining Classes

Each combining character has a normative canonical *combining class*. This class is used with the Canonical Ordering Algorithm to determine which combining characters interact typographically and to determine how the canonical ordering of sequences of combining characters takes place. Class *zero* combining characters act like base letters for the purpose of determining canonical order. Combining characters with non-zero classes participate in reordering for the purpose of determining the canonical order of sequences of characters. (See *Section 3.11, Normalization Forms*, for the specification of the algorithm.)

The list of combining characters and their canonical combining class appears in the Unicode Character Database. Most combining characters are nonspacing.

The canonical order of character sequences does *not* imply any kind of linguistic correctness or linguistic preference for ordering of combining marks in sequences. For more information on rendering combining marks, see *Section 5.13, Rendering Nonspacing Marks*.

Class zero combining marks are never reordered by the Canonical Ordering Algorithm. Except for class zero, the exact numerical values of the combining classes are of no importance in canonical equivalence, although the relative magnitude of the classes is significant. For example, it is crucial that the combining class of the cedilla be lower than the combining class of the dot below, although their exact values of 202 and 220 are not important for implementations.

Certain classes tend to correspond with particular rendering positions relative to the base character, as shown in *Figure 4-1*.

Figure 4-1. Positions of Common Combining Marks
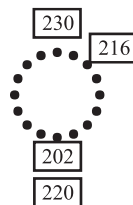
4.3.1 Reordrant, Split, and Subjoined Combining Marks

In some scripts, the rendering of combining marks is notably complex. This is true in particular of the Brahmi-derived scripts of South and Southeast Asia, whose vowels are often encoded as class zero combining marks in the Unicode Standard, known as *matras* for the Indic scripts.

In the case of simple combining marks, as for the accent marks of the Latin script, the normative Unicode combining class of that combining mark typically corresponds to its positional placement with regard to a base letter, as described earlier. However, in the case of the combining marks representing vowels (and sometimes consonants) in the Brahmi-derived scripts and other *abugidas*, all of the combining marks are given the normative combining class of zero, regardless of their positional placement within an *aksara*. The placement and rendering of a class zero combining mark cannot be derived from its combining class alone, but rather depends on having more information about the particulars of the script involved. In some instances, the position may migrate in different historical periods for a script or may even differ depending on font style.

The identification of *matras* in Indic scripts is provided in the data file IndicSyllabicCategory.txt in the Unicode Character Database. Information about their positional placement can be found in the data file IndicPositionalCategory.txt. The following text in this section subcategorizes some of the class zero combining marks for Brahmi-derived scripts, pointing out significant types that need to be handled consistently, and relating their positional placement to the particular values documented in IndicPositionalCategory.txt.

Reordrant Class Zero Combining Marks. In many instances in Indic scripts, a vowel is represented in logical order *after* the consonant of a syllable, but is displayed *before* (to the left of) the consonant when rendered. Such combining marks are termed *reordrant* to reflect their visual reordering to the left of a consonant (or, in some instances, a consonant cluster). Special handling is required for selection and editing of these marks. In particular, the possibility that the combining mark may be reordered to the left side past a cluster, and not simply past the immediate preceding character in the backing store, requires attention to the details for each script involved.

The *visual* reordering of these reordrant class zero combining marks has nothing to do with the reordering of combining character sequences in the Canonical Ordering Algorithm. All of these marks are *class zero* and thus are *never* reordered by the Canonical Ordering Algorithm for normalization. The reordering is purely a presentational issue for *glyphs* during rendering of text.

Reordrant class zero combining marks correspond to the list of characters with Indic_Positional_Category = Left.

In addition, there are historically related vowel characters in the Thai, Lao, New Tai Lue, and Tai Viet scripts that are not treated as combining marks. Instead, for these scripts, such vowels are represented in the backing store in visual order and require no reordering for rendering. The trade-off is that they have to be rearranged for correct sorting. Because of that processing requirement, these characters are given a formal character property assignment, the Logical_Order_Exception property. See

PropList.txt in the Unicode Character Database. The list of characters with the Logical_Order_Exception property is the same as those documented with the value Indic_Positional_Category = Visual_Order_Left in IndicPositionalCategory.txt.

Split Class Zero Combining Marks. In addition to the reordrant class zero combining marks, there are a number of class zero combining marks whose representative glyph typically consists of two parts, which are split into different positions with respect to the consonant (or consonant cluster) in an *aksara*. Sometimes these glyphic pieces are rendered both to the left and the right of a consonant. Sometimes one piece is rendered above or below the consonant and the other piece is rendered to the left or the right. Particularly in the instances where some piece of the glyph is rendered to the left of the consonant, these split class zero combining marks pose similar implementation problems as for the reordrant marks.

The split class zero combining marks have various Indic_Positional_Category values such as Left_And_Right, Top_And_Bottom, Top_And_Right, Top_And_Left, and so forth. See IndicPositionalCategory.txt for the full listing.

One should pay very careful attention to all split class zero combining marks in implementations. Not only do they pose issues for rendering and editing, but they also often have canonical equivalences defined involving the separate pieces, when those pieces are also encoded as characters. As a consequence, the split combining marks may constitute exceptional cases under normalization. Some of the Tibetan split combining marks are deprecated.

The split vowels also pose difficult problems for understanding the standard, as the *phonological* status of the vowel phonemes, the *encoding* status of the characters (including any canonical equivalences), and the *graphical* status of the glyphs are easily confused, both for native users of the script and for engineers working on implementations of the standard.

Subjoined Class Zero Combining Marks. Brahmi-derived scripts that are not represented in the Unicode Standard with a virama may have class zero combining marks to represent subjoined forms of consonants. These correspond graphologically to what would be represented by a sequence of virama plus consonant in other related scripts. The subjoined consonants do not pose particular rendering problems, at least not in comparison to other combining marks, but they should be noted as constituting an exception to the normal pattern in Brahmi-derived scripts of consonants being represented with base letters. This exception needs to be taken into account when doing linguistic processing or searching and sorting.

Subjoined class zero combining marks are listed with the value Indic_Syllabic_Category = Consonant_Subjoined in IndicSyllabicCategory.txt.

Strikethrough Class Zero Combining Marks. The Kharoshthi script is unique in having some class zero combining marks for vowels that are struck through a conso-

nant, rather than being placed in a position around the consonant. These strikethrough combining marks may involve particular problems for implementations. In addition to the Kharoshthi vowels, there are a number of combining *svarita* marks for Vedic texts which are also rendered as overstruck forms. These Kharoshthi vowels and Vedic *svarita* marks have the property value Indic_Positional_Category = Overstruck in IndicPositionalCategory.txt.

4.4 Directionality

Directional behavior is interpreted according to the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").For this purpose, all characters of the Unicode Standard possess a normative *directional* type, defined by the Bidi_Class (bc) property in the Unicode Character Database. The directional types left-to-right and right-to-left are called *strong types*, and characters of these types are called strong directional characters. Left-to-right types include most alphabetic and syllabic characters as well as all Han ideographic characters. Right-to-left types include the letters of predominantly right-to-left scripts, such as Arabic, Hebrew, and Syriac, as well as most punctuation specific to those scripts. In addition, the Unicode Bidirectional Algorithm uses *weak types* and *neutrals*. Interpretation of directional properties according to the Unicode Bidirectional Algorithm is needed for layout of right-to-left scripts such as Arabic and Hebrew.

4.5 General Category

The Unicode Character Database defines a General_Category property for all Unicode code points. The General_Category value for a character serves as a basic classification of that character, based on its primary usage. The property extends the widely used subdivision of ASCII characters into letters, digits, punctuation, and symbols—a useful classification that needs to be elaborated and further subdivided to remain appropriate for the larger and more comprehensive scope of the Unicode Standard.

Each Unicode code point is assigned a normative General_Category value. Each value of the General_Category is given a two-letter property value alias, where the first letter gives information about a major class and the second letter designates a subclass of that major class. In each class, the subclass "other" merely collects the remaining characters of the major class. For example, the subclass "No" (Number, other) includes all characters of the Number class that are not a decimal digit or letter. These characters may have little in common besides their membership in the same major class.

Table 4-4 enumerates the General_Category values, giving a short description of each value. The special values LC, L, M, N, P, S, Z, and C are not part of the enumeration, per se, but instead constitute aliases for closely related sets of values; those aliases for grouped General_Category values are often helpful in regular expressions. See *Table 2-3* for the relationship between General_Category values and basic types of code points.

Table 4-4. General_Category Values

Abbr	Long	Description
Lu	Uppercase_Letter	an uppercase letter
Ll	Lowercase_Letter	a lowercase letter
Lt	Titlecase_Letter	a digraph encoded as a single character, with first part uppercase
LC	Cased_Letter	Lu | Ll | Lt
Lm	Modifier_Letter	a modifier letter
Lo	Other_Letter	other letters, including syllables and ideographs
L	Letter	Lu | Ll | Lt | Lm | Lo
Mn	Nonspacing_Mark	a nonspacing combining mark (zero advance width)
Mc	Spacing_Mark	a spacing combining mark (positive advance width)
Me	Enclosing_Mark	an enclosing combining mark
M	Mark	Mn | Mc | Me
Nd	Decimal_Number	a decimal digit

Nl	Letter_Number	a letterlike numeric character
No	Other_Number	a numeric character of other type
N	Number	Nd | Nl | No
Pc	Connector_Punctuation	a connecting punctuation mark, like a tie
Pd	Dash_Punctuation	a dash or hyphen punctuation mark
Ps	Open_Punctuation	an opening punctuation mark (of a pair)
Pe	Close_Punctuation	a closing punctuation mark (of a pair)
Pi	Initial_Punctuation	an initial quotation mark
Pf	Final_Punctuation	a final quotation mark
Po	Other_Punctuation	a punctuation mark of other type
P	Punctuation	Pc | Pd | Ps | Pe | Pi | Pf | Po
Sm	Math_Symbol	a symbol of mathematical use
Sc	Currency_Symbol	a currency sign
Sk	Modifier_Symbol	a non-letterlike modifier symbol
So	Other_Symbol	a symbol of other type
S	Symbol	Sm | Sc | Sk | So
Zs	Space_Separator	a space character (of various non-zero widths)
Zl	Line_Separator	U+2028 LINE SEPARATOR only
Zp	Paragraph_Separator	U+2029 PARAGRAPH SEPARATOR only
Z	Separator	Zs | Zl | Zp
Cc	Control	a C0 or C1 control code
Cf	Format	a format control character
Cs	Surrogate	a surrogate code point
Co	Private_Use	a private-use character
Cn	Unassigned	a reserved unassigned code point or a noncharacter
C	Other	Cc | Cf | Cs | Co | Cn

There are several other conventions for how General_Category values are assigned to Unicode characters. Many characters have multiple uses, and not all such uses can be captured by a single, simple partition property such as General_Category. Thus, many letters often serve dual functions as numerals in traditional numeral systems. Examples can be found in the Roman numeral system, in Greek usage of letters as numbers, in Hebrew, and similarly for many scripts. In such cases the General_Category is assigned based on the primary letter usage of the character, even though it may also have numeric values, occur in numeric expressions, or be used symbolically in mathematical expressions, and so on.

The General_Category gc=Nl is reserved primarily for letterlike number forms which are not technically digits. For example, the compatibility Roman numeral characters, U+2160..U+217F, all have gc=Nl. Because of the compatibility status of these characters, the recommended way to represent Roman numerals is with regular Latin

letters (gc=Ll or gc=Lu). These letters derive their numeric status from conventional usage to express Roman numerals, rather than from their General_Category value.

Currency symbols (gc=Sc), by contrast, are given their General_Category value based entirely on their function as symbols for currency, even though they are often derived from letters and may appear similar to other diacritic-marked letters that get assigned one of the letter-related General_Category values.

Pairs of opening and closing punctuation are given their General_Category values (gc=Ps for opening and gc=Pe for closing) based on the most typical usage and orientation of such pairs. Occasional usage of such punctuation marks unpaired or in opposite orientation certainly occurs, however, and is in no way prevented by their General_Category values.

Similarly, characters whose General_Category identifies them primarily as a symbol or as a mathematical symbol may function in other contexts as punctuation or even paired punctuation. The most obvious such case is for U+003C "<" LESS-THAN SIGN and U+003E ">" GREATER-THAN SIGN. These are given the General_Category gc = Sm because their primary identity is as mathematical relational signs. However, as is obvious from HTML and XML, they also serve ubiquitously as paired bracket punctuation characters in many formal syntaxes.

A common use of the General_Category of a Unicode character is in the derivation of properties for the determination of text boundaries, as in Unicode Standard Annex #29, "Unicode Text Segmentation." Other common uses include determining language identifiers for programming, scripting, and markup, as in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax," and in regular expression languages such as Perl. For more information, see Unicode Technical Standard #18, "Unicode Regular Expressions."

This property is also used to support common APIs such as `isDigit()`. Common functions such as `isLetter()` and `isUppercase()` do not extend well to the larger and more complex repertoire of Unicode. While it is possible to naively extend these functions to Unicode using the General_Category and other properties, they will not work for the entire range of Unicode characters and the kinds of tasks for which people intend them. For more appropriate approaches, see Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax"; Unicode Standard Annex #29, "Unicode Text Segmentation"; *Section 5.18, Case Mappings*; and *Section 4.10, Letters, Alphabetic, and Ideographic*.

Although the General_Category property is normative, and its values are used in the derivation of many other properties referred to by Unicode algorithms, it does not follow that the General_Category always provides the most appropriate classification of a character for any given purpose. Implementations are not required to treat characters solely according to their General_Category values when classifying them in

various contexts. The following examples illustrate some typical cases in which an implementation might reasonably diverge from General_Category values for a character when grouping characters as "punctuation," "symbols," and so forth.

- A character picker application might classify U+0023 # NUMBER SIGN among symbols, or perhaps under both symbols and punctuation.

- An "Ignore Punctuation" option for a search might choose not to ignore U+0040 @ COMMERCIAL AT.

- A layout engine might treat U+0021 ! EXCLAMATION MARK as a mathematical operator in the context of a mathematical equation, and lay it out differently than if the same character were used as terminal punctuation in text.

- Japanese *wakiten* (脇点) is a style of text emphasis which uses a sideline sequence of "dots" (of different types and shapes) to highlight Japanese text in vertical layout (or above text in horizontal layout). Implementations of *wakiten* follow a general rule that in the emphasized text span, the dots are applied to "letters" and "symbols," but are not applied to "punctuation." In most cases the definition of punctuation used in *wakiten* matches the Unicode General_Category, but the following common characters [# % & @ § ¶] are considered to be symbols, rather than punctuation.

- A regular expression syntax could provide an operator to match all punctuation, but include characters other than those limited to gc = P (for example, U+00A7 § SECTION SIGN).

The general rule is that if an implementation *purports* to be using the Unicode General_Category property, then it must use the exact values specified in the Unicode Character Database for that claim to be conformant. Thus, if a regular expression syntax explicitly supports the Unicode General_Category property and matches gc = P, then that match must be based on the precise UCD values.

4.6 Numeric Value

Numeric_Value and Numeric_Type are normative properties of characters that represent *numbers*. Characters with a non-default Numeric_Type include numbers and number forms such as fractions, subscripts, superscripts, Roman numerals, encircled numbers, and many script-specific digits and numbers.

In some traditional numbering systems, ordinary letters may also be used with a numeric value. Examples include Greek letters used numerically, Hebrew gematria, and even Latin letters when used in outlines (II.A.1.b). Letter characters used in this way are not given Numeric_Type or Numeric_Value property values, to prevent simplistic parsers from treating them numerically by mistake. The Unicode Character Database gives the Numeric_Type and Numeric_Value property values only for Unicode characters that *normally* represent numbers.

Decimal Digits. Decimal digits, as commonly understood, are digits used to form decimal-radix numbers. They include script-specific digits, but exclude characters such as Roman numerals and Greek acrophonic numerals, which do not form decimal-radix expressions. (Note that <1, 5> = 15 = fifteen, but <I, V> = IV = four.)

The Numeric_Type = Decimal property value (which is correlated with the General_Category = Nd property value) is limited to those numeric characters that are used in decimal-radix numbers and for which a full set of digits has been encoded in a contiguous range, with ascending order of Numeric_Value, and with the digit zero as the first code point in the range.

Decimal digits, as defined in the Unicode Standard by these property assignments, exclude some characters, such as the CJK ideographic digits (see the first ten entries in *Table 4-5*), which are not encoded in a contiguous sequence. Decimal digits also exclude the compatibility subscript and superscript digits, to prevent simplistic parsers from misinterpreting their values in context. (For more information on superscript and subscripts, see *Section 22.4, Superscript and Subscript Symbols*.) Traditionally, the Unicode Character Database has given these sets of noncontiguous or compatibility digits the value Numeric_Type = Digit, to recognize the fact that they consist of digit values but do not necessarily meet all the criteria for Numeric_Type = Decimal. However, the distinction between Numeric_Type = Digit and the more generic Numeric_Type = Numeric has proven not to be useful in implementations. As a result, future sets of digits which may be added to the standard and which do not meet the criteria for Numeric_Type = Decimal will simply be assigned the value Numeric_Type = Numeric.

Numbers other than decimal digits can be used in numerical expressions, and may be interpreted by a numeric parser, but it is up to the implementation to determine such specialized uses.

Script-Specific Digits. The Unicode Standard encodes separate characters for the digits specific to a given script. Examples are the digits used with the Arabic script or those of the various Indic scripts. See *Table 22-3* for a list of script-specific digits. For naming conventions relevant to the Arabic digits, see the introduction to *Section 9.2, Arabic*.

4.6.1 Ideographic Numeric Values

CJK ideographs also may have numeric values. The primary numeric ideographs are shown in *Table 4-5*. The Numeric_Value property for each of them is shown in the second column of the table. In a few instances, there is regional variation in values, with one value currently used in China and another used in Japan. In such cases, as for 兆, which is used for *million* (zhào) in China, but for *trillion* (chō) in Japan, both values are listed for kPrimaryNumeric in the Unihan database, but the first of those values is used to calculate Numeric_Value in the Unicode Character Database.

When used to represent numbers in decimal notation, zero is represented by U+3007. Otherwise, zero is represented by U+96F6.

Table 4-5. Primary Numeric Ideographs

Code Point	Numeric_Value	Secondary Value
U+96F6	0	
U+4E00	1	
U+4E8C	2	
U+4E09	3	
U+56DB	4	
U+4E94	5	
U+516D	6	
U+4E03	7	
U+516B	8	
U+4E5D	9	
U+5341	10	
U+767E	100	
U+5343	1,000	
U+4E07	10,000	
U+842C	10,000	
U+5146	1,000,000	1,000,000,000,000 (10,000 × 10,000 × 10,000)
U+79ED	1,000,000	1,000,000,000,000 (10,000 × 10,000 × 10,000)
U+5104	100,000,000 (10,000 × 10,000)	
U+4EBF	100,000,000 (10,000 × 10,000)	
U+4EAC	10,000,000,000,000	

Ideographic accounting numbers are commonly used on checks and other financial instruments to minimize the possibilities of misinterpretation or fraud in the representation of numerical values. The set of accounting numbers varies somewhat between Japanese, Chinese, and Korean usage. *Table 4-6* gives a fairly complete listing of the known accounting characters. Some of these characters are ideographs with other meanings pressed into service as accounting numbers; others are used only as accounting numbers.

Table 4-6. Ideographs Used as Accounting Numbers

Number	Multiple Uses	Accounting Use Only
1	U+58F9, U+58F1	U+5F0C
2		U+8CAE, U+8CB3, U+8D30, U+5F10, U+5F0D
3	U+53C3, U+53C2	U+53C1, U+5F0E
4	U+8086	
5	U+4F0D	
6	U+9678, U+9646	
7	U+67D2	
8	U+634C	
9	U+7396	
10	U+62FE	
100	U+964C	U+4F70
1,000	U+4EDF	
10,000	U+842C	

In Japan, U+67D2 is also pronounced *urusi*, meaning "lacquer," and is treated as a variant of the standard character for "lacquer," U+6F06.

The Unihan Database gives the most up-to-date and complete listing of primary numeric ideographs and ideographs used as accounting numbers, including those for CJK repertoire extensions beyond the Unified Repertoire and Ordering. See Unicode Standard Annex #38, "Unicode Han Database (Unihan)," for more details.

4.7 Bidi Mirrored

Bidi Mirrored is a normative property of characters such as parentheses, whose images are mirrored horizontally in text that is laid out from right to left. For example, U+0028 LEFT PARENTHESIS is interpreted as *opening parenthesis*; in a left-to-right context it will appear as "(", while in a right-to-left context it will appear as the mirrored glyph ")". This requirement is necessary to render the character properly in a bidirectional context. Mirroring is the default behavior for such paired characters in Unicode text. (For more information, see the "Paired Punctuation" subsection in *Section 6.2, General Punctuation*.)

Paired delimiters are mirrored even when they are used in unusual ways, as, for example, in the mathematical expressions [a,b) or]a,b[. If any of these expressions is displayed from right to left, then the mirrored glyphs are used. Because of the difficulty in interpreting such expressions, authors of bidirectional text need to make sure that readers can determine the desired directionality of the text from context.

Note that mirroring is not limited to paired punctuation and other paired delimiters, but also applies to a limited set of mathematical symbols whose orientation is reversed when the direction of line layout is reversed—for example, U+222B INTEGRAL. Such characters subject to bidi mirroring require the availability of a left-right symmetric pair of glyphs for correct display.

For some mathematical symbols, the "mirrored" form is not an exact mirror image. For example, the direction of the circular arrow in U+2232 CLOCKWISE CONTOUR INTEGRAL reflects the direction of the integration in coordinate space, not the text direction. In a right-to-left context, the integral sign would be mirrored, but the circular arrow would retain its direction. In a similar manner, the bidi-mirrored form of U+221B CUBE ROOT would be composed of a mirrored radix symbol with a non-mirrored digit "3". For more information, see Unicode Technical Report #25, "Unicode Support for Mathematics."

The list of mirrored characters appears in the Unicode Character Database. Formally, they consist of all characters with the property value Bidi_Mirrored = Y. This applies to almost all paired brackets (with the legacy exception of U+FD3E ORNATE LEFT PARENTHESIS and U+FD3F ORNATE RIGHT PARENTHESIS), but not to quotation marks, whose directionality and pairing status is less predictable than paired brackets. (See the subsection on "Language-Based Usage of Quotation Marks" in *Section 6.2, General Punctuation*.) Many mathematical operators with a directional orientation are bidi mirrored, but mirroring does *not* apply to any arrow symbols.

The mirroring behavior noted in paleographic materials for a number of ancient scripts, such as Old Italic, Runic, (ancient) Greek, Egyptian Hieroglyphs, and so forth, is *not* within the scope of the Bidi Mirrored property, and is not handled by

default in the Unicode Bidirectional Algorithm (UBA). Mirroring of the letters or signs in the text of such paleographic material should be dealt with by higher level protocol. HL6 "Additional mirroring" is specified by the UBA as a permissible type of higher-level protocol to allow additional mirroring of glyphs for certain characters in a bidirectional context. A straightforward approach to a higher-level protocol would use existing bidirectional format controls to override text layout direction, add mirrored glyphs to a font used for paleographic display, and make the display choice depend on resolved direction for a directional run. HL3 "Emulate explicit directional formatting characters" in the UBA also allows a higher-level protocol to use other techniques such as style sheets or markup to override text directionality in structured text. In combination, such techniques can provide for the layout requirements of paleographic scripts which may mirror letters or signs depending on text layout direction. See the discussions of directionality and text layout in the respective sections regarding each script.

Related Properties. The *Bidi Mirrored* property is not to be confused with the related, informative *Bidi Mirroring Glyph* property, which lists pairs of characters whose representative glyphs are mirror images of each other. The Unicode Bidirectional Algorithm also requires two related, normative properties, *Bidi Paired Bracket* and *Bidi Paired Bracket Type*, which are used for matching specific bracket pairs and to assign the same text direction to both members of each pair in bidirectional processing for text layout. These properties do not affect mirroring. For more information, see BidiMirroring.txt and BidiBrackets.txt in the Unicode Character Database.

4.8 Name

Unicode characters have names that serve as unique identifiers for each character. The character names in the Unicode Standard are identical to those of the English-language edition of ISO/IEC 10646.

Where possible, character names are derived from existing conventional names of a character or symbol in English, but in many cases the character names nevertheless differ from traditional names widely used by relevant user communities. The character names of symbols and punctuation characters often describe their shape, rather than their function, because these characters are used in many different contexts. See also "Color Words in Unicode Character Names" in <u>Section 22.9, Miscellaneous Symbols</u>.

Character names are listed in the code charts. Currently, one of the characters with the longest name is U+1FBA8 BOX DRAWINGS LIGHT DIAGONAL UPPER CENTRE TO MIDDLE LEFT AND MIDDLE RIGHT TO LOWER CENTRE (Version 13.0) with 88 letters and spaces in its name, and the one with the shortest name is U+1F402 OX (Version 6.0) with only two letters in its name.

Stability. Once assigned, a character name is immutable. It will never be changed in subsequent versions of the Unicode Standard. Implementers and users can rely on the fact that a character name uniquely represents a given character.

Character Name Syntax. Unicode character names, as listed in the code charts, contain only uppercase Latin letters A through Z, digits, space, and hyphen-minus. In more detail, character names reflect the following rules:

- **R1** ***Only Latin capital letters A to Z (U+0041..U+005A), ASCII digits (U+0030.. U+0039),*** U+0020 SPACE***, and*** U+002D HYPHEN-MINUS ***occur in character names.***

- **R2** ***Digits do not occur as the first character of a character name, nor immediately following a space character.***

- **R3** U+002D HYPHEN-MINUS ***does not occur as the first or last character of a character name, nor immediately between two spaces, nor immediately preceding or following another hyphen-minus character. (In other words, multiple occurrences of U+002D in sequence are not allowed.)***

- **R4** ***A space does not occur as the first or last character of a character name, nor immediately preceding or following another space character. (In other words, multiple spaces in sequence are not allowed.)***

See *Appendix A, Notational Conventions*, for the typographical conventions used when printing character names in the text of the standard.

Names as Identifiers. Character names are constructed so that they can easily be transposed into formal identifiers in another context, such as a computer language. Because Unicode character names do not contain any underscore ("_") characters, a common strategy is to replace any *hyphen-minus* or space in a character name by a single "_" when constructing a formal identifier from a character name. This strategy automatically results in a syntactically correct identifier in most formal languages. Furthermore, such identifiers are guaranteed to be unique, because of the special rules for character name matching.

Character Name Matching. When matching identifiers transposed from character names, it is possible to ignore case, whitespace, and all medial *hyphen-minus* characters (or any "_" replacing a *hyphen-minus*), except for the *hyphen-minus* in U+1180 HANGUL JUNGSEONG O-E, and still result in a unique match. For example, "ZERO WIDTH SPACE" is equivalent to "zero-width-space" or "ZERO_WIDTH_SPACE" or "ZeroWidthSpace". However, "TIBETAN LETTER A" should not match "TIBETAN LETTER -A", because in that instance the *hyphen-minus* is not medial between two letters, but is instead preceded by a space. For more information on character name matching, see Section 5.9, "Matching Rules" in Unicode Standard Annex #44, "Unicode Character Database."

Named Character Sequences. Occasionally, character sequences are also given a normative name in the Unicode Standard. The names for such sequences are taken from the same namespace as character names, and are also unique. For details, see Unicode Standard Annex #34, "Unicode Named Character Sequences." Named character sequences are not listed in the code charts; instead, they are listed in the file NamedSequences.txt in the Unicode Character Database.

The names for named character sequences are also immutable. Once assigned, they will never be changed in subsequent versions of the Unicode Standard.

Character Name Aliases. The Unicode Standard has a mechanism for the publication of additional, normative formal aliases for characters. These formal aliases are known as *character name aliases*. (See Definition D5 in *Section 3.3, Semantics*.) They function essentially as auxiliary names for a character. The original reason for defining character name aliases was to provide corrections for known mistakes in character names, but they have also proven useful for other purposes, as documented here.

Character name aliases are listed in the file NameAliases.txt in the Unicode Character Database. That file also documents the type field which distinguishes among different kinds of character name aliases, as shown in *Table 4-7*.

Table 4-7. Types of Character Name Aliases

Type	Description
correction	Corrections for serious problems in the character names
control	ISO 6429 names for C0 and C1 control functions, and other commonly occurring names for control codes
alternate	Widely used alternate names for format characters
figment	Several documented labels for C1 control code points which were never actually approved in any standard
abbreviation	Commonly occurring abbreviations (or acronyms) for control codes, format characters, spaces, and variation selectors

Character name aliases are immutable, once published. (See Definition D42 in *Section 3.5, Properties*.) They follow the same syntax rules as character names and are also guaranteed to be unique in the Unicode namespace for character names. This attribute makes character name aliases useful as identifiers. A character may, in principle, have more than one normative character name alias, but each distinct character name alias uniquely identifies only a single code point.

The first type of character name alias consists of corrections for known mistakes in character names. Sometimes errors in a character name are only discovered after publication of a version of the Unicode Standard. Because character names are immutable, such errors are not corrected by changing the names after publication. However, in some limited instances (as for obvious typos in the name), a character name alias is defined instead.

For example, the following Unicode character name has a well-known spelling error in it:

U+FE18 PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRAKCET

Because the spelling error could not be corrected after publication of the data files which first contained it, a character name alias with the corrected spelling was defined:

U+FE18 PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRACKET

Character name aliases are provided for additional reasons besides corrections of errors in the character names. For example, there are character name aliases which give definitive labels to control codes, which have no actual Unicode character names:

U+0009 HORIZONTAL TABULATION

Character name aliases of type *alternate* are for widely used alternate names of Unicode format characters. Currently only one such alternate is normatively defined, but it is for an important character:

    U+FEFF BYTE ORDER MARK

Among the control codes there are a few which have had names propagate through the computer implementation "lore," despite the fact that they refer to ISO/IEC 10646 control functions that were never formally adopted. These names are defined as character name aliases of type *figment*, and are included in NameAliases.txt, because they occur in some widely distributed implementations, such as the regex engine for Perl. Examples include:

    U+0081 HIGH OCTET PRESET

Additional character name aliases match existing and widely used abbreviations (or acronyms) for control codes and for Unicode format characters:

    U+0009 TAB

    U+200B ZWSP

Specifying these additional, normative character name aliases serves two major functions. First, it provides a set of well-defined aliases for use in regular expression matching and searching, where users might expect to be able to use established names or abbreviations for control codes and the like, but where those names or abbreviations are not part of the actual Unicode Name property. Second, because character name aliases are guaranteed to be unique in the Unicode character name namespace, having them defined for control codes and abbreviations prevents the potential for accidental collisions between de facto current use and names which might be chosen in the future for newly encoded Unicode characters.

It is acceptable and expected for external specifications to make normative references to Unicode characters using one (or more) of their normative character name aliases, where such references make sense. For example, when discussing Unicode encoding schemes and the role of U+FEFF as a signature for byte order, it would not make much sense to insist on referring to U+FEFF by its name ZERO WIDTH NO-BREAK SPACE, when use of the character name alias BYTE ORDER MARK or the widely used abbreviation BOM would communicate with less confusion.

A subset of character name aliases is listed in the code charts, using special typographical conventions explained in *Section 24.1, Character Names List*.

A normative character name alias is distinct from the informative aliases listed in the code charts. Informative aliases merely point out other common names in use for a given character. Informative aliases are not immutable and are not guaranteed to be

unique; they therefore cannot serve as an identifier for a character. Their main purposes are to help readers of the standard to locate and to identify particular characters.

4.8.1 Unicode Name Property

Formally, the character name for a Unicode character is the value of the normative character property, "Name". Most Unicode character properties are defined by enumeration in one of the data files of the Unicode Character Database, but the Name property is instead defined in part by enumeration and in part by rule. A significant proportion of Unicode characters belong to large sets, such as Han ideographs, Tangut ideographs, and Hangul syllables, for which the character names are best defined by generative rule, rather than one-by-one naming.

Formal Definition of the Name Property. The Name property (short alias: "na") is a string property, defined as follows:

*NR1 **For Hangul syllables, the Name property value is derived by rule, as specified in <u>Section 3.12, Conjoining Jamo Behavior</u>, under "Hangul Syllable Name Generation," by concatenating a fixed prefix string "HANGUL SYLLABLE " and appropriate values of the Jamo_Short_Name property.***

For example, the name of U+D4DB is HANGUL SYLLABLE PWILH, constructed by concatenation of "HANGUL SYLLABLE " and three Jamo_Short_Name property values, "P" + "WI" + "LH".

*NR2 **For most ideographs (characters with the binary property value Ideographic = True), the Name property value is derived by concatenating a script-specific prefix string, as specified in <u>Table 4-8</u>, to the code point, expressed in uppercase hexadecimal, with the usual 4- to 6-digit convention.***

The exact specification of the 4-6 digit hexadecimal convention for expressing Unicode code points can be found in *<u>Appendix A, Notational Conventions</u>*.

For example, the name of U+4E00 is CJK UNIFIED IDEOGRAPH-4E00, constructed by concatenation of "CJK UNIFIED IDEOGRAPH-" and the code point. Similarly, the character name of U+17000 is TANGUT IDEOGRAPH-17000.

*NR3 **For all other Graphic characters and for all Format characters, the Name property value is as explicitly listed in Field 1 of UnicodeData.txt.***

For example, U+0A15 GURMUKHI LETTER KA or U+200D ZERO WIDTH JOINER.

NR4 **For all other Unicode code points of all other types (Control, Private-Use, Surrogate, Noncharacter, and Reserved), the value of the Name property is the null string. In other words, na = "".**

The ranges of Hangul syllables and most Han and Tangut ideographic characters subject to the name derivation rules NR1 and NR2 are identified by a special convention in Field 1 of UnicodeData.txt. The start and end of each range are indicated by a pair of entries in the data file in the general format:

```
NNNN;<RANGENAME, First>;Lo;0;L;;;;;N;;;;;
NNNN;<RANGENAME, Last>;Lo;0;L;;;;;N;;;;;
```

This convention originated as a compression technique for UnicodeData.txt, as all of the UnicodeData.txt properties of these ranges were uniform, and the names for the characters in the ranges could be specified by rule. Note that the same convention is used in UnicodeData.txt to specify properties for code point types which have a null string as their Name property value, such as private use characters.

CJK compatibility ideographs are an exception. They have names derived by rule NR2, but are *explicitly* listed in UnicodeData.txt with their names, because they typically have non-uniform character properties, including most notably a nontrivial canonical decomposition value. Ideographic characters for scripts other than Han and Tangut, as well as Egyptian hieroglyphs, also have their names listed explicitly in UnicodeData.txt, even when their names are derived by rule NR2.

The exact ranges subject to name derivation rules NR1 and NR2, and the specified prefix strings are summarized in *Table 4-8*.

Table 4-8. Name Derivation Rule Prefix Strings

Range	Rule	Prefix String
AC00..D7A3	NR1	"HANGUL SYLLABLE "
3400..4DBF	NR2	"CJK UNIFIED IDEOGRAPH-"
4E00..9FFF	NR2	"CJK UNIFIED IDEOGRAPH-"
20000..2A6DF	NR2	"CJK UNIFIED IDEOGRAPH-"
2A700..2B739	NR2	"CJK UNIFIED IDEOGRAPH-"
2B740..2B81D	NR2	"CJK UNIFIED IDEOGRAPH-"
2B820..2CEA1	NR2	"CJK UNIFIED IDEOGRAPH-"
2CEB0..2EBE0	NR2	"CJK UNIFIED IDEOGRAPH-"
2EBF0..2EE5D	NR2	"CJK UNIFIED IDEOGRAPH-"
30000..3134A	NR2	"CJK UNIFIED IDEOGRAPH-"
31350..323AF	NR2	"CJK UNIFIED IDEOGRAPH-"
13460..143FA	NR2	"EGYPTIAN HIEROGLYPH-"
17000..187F7	NR2	"TANGUT IDEOGRAPH-"

18D00..18D08	NR2	"TANGUT IDEOGRAPH-"
18B00..18CD5	NR2	"KHITAN SMALL SCRIPT CHARACTER-"
1B170..1B2FB	NR2	"NUSHU CHARACTER-"
F900..FA6D*	NR2	"CJK COMPATIBILITY IDEOGRAPH-"
FA70..FAD9	NR2	"CJK COMPATIBILITY IDEOGRAPH-"
2F800..2FA1D	NR2	"CJK COMPATIBILITY IDEOGRAPH-"

Twelve of the CJK ideographs in the starred range in *Table 4-8*, in the CJK Compatibility Ideographs block, are actually CJK unified ideographs. Nonetheless, their names are constructed with the "CJK COMPATIBILITY IDEOGRAPH-" prefix shared by all other code points in that block. The status of a CJK ideograph as a unified ideograph cannot be deduced from the Name property value for that ideograph; instead, the dedicated binary property Unified_Ideograph should be used to determine that status. See "CJK Compatibility Ideographs" in *Section 18.1, Han*, and Section 4.4, "Listing of Characters Covered by the Unihan Database" in Unicode Standard Annex #38, "Unihan Database," for more details about these exceptional twelve CJK ideographs.

The generic term "character name" refers to the Name property value for an encoded Unicode character. An expression such as, "The reserved code point U+1029F has no name," is shorthand for the more precise statement that the reserved code point U+1029F (as for all code points of type Reserved) has a property value of na = "" for the Name property.

Name Uniqueness. The Unicode Name property values are unique for all non-null values, but not every Unicode code point has a unique Unicode Name property value. Furthermore, because Unicode character names, character name aliases, named character sequences, and code point labels constitute a single, unique namespace, the Name property value uniqueness requirement applies to all three kinds of names and to code point labels.

Interpretation of Field 1 of UnicodeData.txt. Where Field 1 of UnicodeData.txt contains a string enclosed in angle brackets, "<" and ">", such a string is *not* a character name, but a meta-label indicating some other information—for example, the start or end of a character range. In these cases, the Name property value for that code point is either empty (na = "") or is given by one of the rules described above. In all *other* cases, the value of Field 1 (that is, the string of characters between the first and second semicolon separators on each line) corresponds to the normative value of the Name property for that code point.

Control Codes. The Unicode Standard does not define character names for control codes (characters with General_Category = Cc). In other words, all control codes have a property value of na = "" for the Name property. Control codes are instead listed in UnicodeData.txt with a special label "<control>" in Field 1. This value is not

a character name, but instead indicates the *code point type* (see Definition D10a in *Section 3.4, Characters and Encoding*). For control characters, the values of the *informative* Unicode 1.0 name property (Unicode_1_Name) in Field 10 match the names of the associated control functions from ISO/IEC 6429. (See *Section 4.9, Unicode 1.0 Names*.)

4.8.2 Code Point Labels

To provide unique, meaningful labels for code points that do not have character names, the Unicode Standard uses a convention for code point labeling.

For each code point type without character names, code point labels are constructed by using a lowercase prefix derived from the code point type, followed by a *hyphen-minus* and then a 4- to 6-digit hexadecimal representation of the code point. Leading zeros are omitted in the hexadecimal representation, unless their omission would result in fewer than four hexadecimal digits. Thus, for example, the code point label for U+0008 would be uniquely constructed as "control-0008", and not as "control-8" or "control-00008" or "control-000008". The label construction for the five affected code point types is illustrated in *Table 4-9*.

Table 4-9. Construction of Code Point Labels

Type	Label
Control	`control-NNNN`
Reserved	`reserved-NNNN`
Noncharacter	`noncharacter-NNNN`
Private-Use	`private-use-NNNN`
Surrogate	`surrogate-NNNN`

Unicode code point labels are included in the unique namespace for Unicode character names. This ensures that there will never be a naming conflict between a code point label and an actual, assigned Unicode character name.

To avoid any possible confusion with actual, non-null Name property values, constructed Unicode code point labels are often displayed between angle brackets: `<control-0009>`, `<noncharacter-FFFF>`, and so on. This convention is used consistently in the data files for the Unicode Character Database.

A constructed code point label is distinguished from the designation of the code point itself (for example, "U+0009" or "U+FFFF"), which is also a unique identifier, as described in *Appendix A, Notational Conventions*.

4.8.3 Use of Character Names in APIs and User Interfaces

Use in APIs. APIs which return the value of a Unicode "character name" for a given code point might vary somewhat in their behavior. An API which is defined as *strictly* returning the value of the Unicode Name property (the "na" attribute), should return a null string for any Unicode code point other than graphic or format characters, as that is the actual value of the property for such code points. On the other hand, an API which returns a name for Unicode code points, but which is expected to provide useful, unique labels for unassigned, reserved code points and other special code point types, should return the value of the Unicode Name property for any code point for which it is non-null, but should otherwise construct a code point label to stand in for a character name.

User Interfaces. A list of Unicode character names may not always be the most appropriate set of choices to present to a user in a user interface. Many common characters do not have a single name for all English-speaking user communities and, of course, their native name in another language is likely to be different altogether. The names of many characters in the Unicode Standard are based on specific Latin transcription of the sounds they represent. There are often competing transcription schemes. For all these reasons, it can be more effective for a user interface to use names that were translated or otherwise adjusted to meet the expectations of the targeted user community. By also listing the formal character name, a user interface could ensure that users can unambiguously refer to the character by the name documented in the Unicode Standard.

4.9 Unicode 1.0 Names

The Unicode_1_Name property is an informative property referring to the name of characters in Version 1.0 of the Unicode Standard. Values of the Unicode_1_Name property are provided in UnicodeData.txt in the Unicode Character Database in cases where the Version 1.0 name of a character differed from the current name of that character. A significant number of names for Unicode characters in Version 1.0 were changed during the process of merging the repertoire of the Unicode Standard with ISO/IEC 10646 in 1991. Character name changes are now strictly prohibited by the Unicode Character Encoding Stability Policy, and no character name has been changed since Version 2.0.

The Version 1.0 names are primarily of historic interest regarding the early development of the Unicode Standard. However, where a Version 1.0 character name provides additional useful information about the identity of a character, it is explicitly listed in the code charts. For example, U+00B6 PILCROW SIGN has its Version 1.0 name, PARAGRAPH SIGN, listed for clarity.

The status of the Unicode_1_Name property values in the case of control codes differs from that for other characters. *The Unicode Standard, Version 1.0,* gave names to the C0 control codes, U+0000..U+001F, U+007F, based on then-current practice for reference to ASCII control codes. Unicode 1.0 gave no names to the C1 control codes, U+0080..U+009F. The values of the Unicode_1_Name property have been updated for the control codes to reflect the ISO/IEC 6429 standard names for control functions. Those names can be seen as annotations in the code charts. In a few instances, because of updates to ISO/IEC 6429, those names may differ from the names that actually occurred in Unicode 1.0. For example, the Unicode 1.0 name of U+0009 was HORIZONTAL TABULATION, but the ISO/IEC 6429 name for this function is CHARACTER TABULATION, and the commonly used alias is, of course, merely *tab*.

4.10 Letters, Alphabetic, and Ideographic

Letters and Syllables. The concept of a letter is used in many contexts. Computer language standards often characterize identifiers as consisting of letters, syllables, ideographs, and digits, but do not specify exactly what a "letter," "syllable," "ideograph," or "digit" is, leaving the definitions implicitly either to a character encoding standard or to a locale specification. The large scope of the Unicode Standard means that it includes many writing systems for which these distinctions are not as self-evident as they may once have been for systems designed to work primarily for Western European languages and Japanese. In particular, while the Unicode Standard includes various "alphabets" and "syllabaries," it also includes writing systems that fall somewhere in between. As a result, no attempt is made to draw a sharp property distinction between letters and syllables.

Alphabetic. The Alphabetic property is a derived informative property of the primary units of alphabets and/or syllabaries, whether combining or noncombining. Included in this group would be composite characters that are canonical equivalents to a combining character sequence of an alphabetic base character plus one or more combining characters; letter digraphs; contextual variants of alphabetic characters; ligatures of alphabetic characters; contextual variants of ligatures; modifier letters; letterlike symbols that are compatibility equivalents of single alphabetic letters; and miscellaneous letter elements. Notably, U+00AA FEMININE ORDINAL INDICATOR and U+00BA MASCULINE ORDINAL INDICATOR are simply abbreviatory forms involving a Latin letter and should be considered alphabetic rather than nonalphabetic symbols.

Ideographic. The Ideographic property is an informative property defined in the Unicode Character Database. The Ideographic property is used, for example, in determining line breaking behavior. Characters with the Ideographic property include unified CJK ideographs, CJK compatibility ideographs, Tangut ideographs, Nüshu ideographs, and characters from other blocks—for example, U+3007 IDEOGRAPHIC NUMBER ZERO and U+3006 IDEOGRAPHIC CLOSING MARK. For more information about Han, Tangut, and Nüshu ideographs, see *Section 18.1, Han*, *Section 18.11, Tangut* and *Section 18.8, Nüshu*. For more about ideographs and logosyllabaries in general, see *Section 6.1, Writing Systems*.

4.11 Properties for Text Boundaries

The determination of text boundaries, such as word breaks or line breaks, involves contextual analysis of potential break points and the characters that surround them. Such an analysis is based on the classification of all Unicode characters by their default interaction with each particular type of text boundary. For example, the Line_Break property defines the default behavior of Unicode characters with respect to line breaking.

A number of characters have special behavior in the context of determining text boundaries. These characters are described in more detail in the subsection on "Line and Word Breaking" in *Section 23.2, Layout Controls*. For more information about text boundaries and these characters, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm," and Unicode Standard Annex #29, "Unicode Text Segmentation."

4.12 Characters with Unusual Properties

The behavior of most characters does not require special attention in this standard. However, the characters in *Table 4-10* exhibit special behavior. Many other characters behave in special ways but are not noted here, either because they do not affect surrounding text in the same way or because their use is intended for well-defined contexts. Examples include the compatibility characters for block drawing, the symbol pieces for large mathematical operators, and many punctuation symbols that need special handling in certain circumstances. Such characters are more fully described in the following chapters. The section numbers or other references listed in the "Details" column in *Table 4-10* indicate where to find more information about the functions or particular groups of characters listed.

Table 4-10. Unusual Properties

Function	Details	Code Point and Name
Segmentation		
Line break controls	*Section 23.2*	00AD SOFT HYPHEN 200B ZERO WIDTH SPACE 2060 WORD JOINER
Combining Marks		
Bases for display of isolated nonspacing marks	*Section 2.11*, *Section 6.2*, *Section 23.2*	0020 SPACE 00A0 NO-BREAK SPACE
Double nonspacing marks	*Section 7.9*	035C COMBINING DOUBLE BREVE BELOW 035D COMBINING DOUBLE BREVE 035E COMBINING DOUBLE MACRON 035F COMBINING DOUBLE MACRON BELOW 0360 COMBINING DOUBLE TILDE 0361 COMBINING DOUBLE INVERTED BREVE 0362 COMBINING DOUBLE RIGHTWARDS ARROW BELOW 1DCD COMBINING DOUBLE CIRCUMFLEX ABOVE 1DFC COMBINING DOUBLE INVERTED BREVE BELOW
Combining half marks	*Section 7.9*	FE20 COMBINING LIGATURE LEFT HALF FE21 COMBINING LIGATURE RIGHT HALF and all other pairs in the Combining Half Marks block
Combining continuous lining marks	*Section 7.3*, *Section 7.9*	0305 COMBINING OVERLINE 0332 COMBINING LOW LINE 0333 COMBINING DOUBLE LOW LINE 033F COMBINING DOUBLE OVERLINE FE26 COMBINING CONJOINING MACRON FE2D COMBINING CONJOINING MACRON BELOW
Combining marks with nondefault stacking	*Section 7.9*	1ABB COMBINING PARENTHESES ABOVE 1ABC COMBINING DOUBLE PARENTHESES ABOVE 1ABD COMBINING PARENTHESES BELOW

Ligation		
Cursive joining and ligation control	*Section 23.2*	200C ZERO WIDTH NON-JOINER 200D ZERO WIDTH JOINER
Fraction formatting	*Section 6.2*	2044 FRACTION SLASH
Ligating modifier tone letters	*Section 7.8*	02E5..02E9 MODIFIER LETTER EXTRA-HIGH TONE BAR..MODIFIER LETTER EXTRA-LOW TONE BAR A712..A716 MODIFIER LETTER EXTRA-HIGH LEFT-STEM TONE BAR..MODIFIER LETTER EXTRA-LOW LEFT-STEM TONE BAR
Ligating brackets that surround text	*Section 11.4*, *Section 13.4*	0F3C TIBETAN MARK ANG KHANG GYON 0F3D TIBETAN MARK ANG KHANG GYAS 13258..1325D EGYPTIAN HIEROGLYPH O006A..EGYPTIAN HIEROGLYPH O006F 13282 EGYPTIAN HIEROGLYPH O033A 13286..13289 EGYPTIAN HIEROGLYPH O036A..EGYPTIAN HIEROGLYPH O036D 13379..1337B EGYPTIAN HIEROGLYPH V011A..EGYPTIAN HIEROGLYPH V011C
Ligating regional indicator symbols	*Section 22.10*, UTS #51	1F1E6..1F1FF REGIONAL INDICATOR SYMBOL LETTER A..REGIONAL INDICATOR SYMBOL LETTER Z
Indic-related: conjuncts, killers, and other viramas		
Brahmi-derived script dead-character formation	*Chapter 12*, *Chapter 13*, *Chapter 14*, *Chapter 15*, *Chapter 16*	See IndicSyllabicCategory.txt in the UCD for a full listing.
Brahmi number formation	*Section 14.1*	1107F BRAHMI NUMBER JOINER
Non-Indic consonant ligation	*Section 19.3*	2D7F TIFINAGH CONSONANT JOINER
Historical viramas with other functions	*Section 13.4*, *Section 13.6*, *Section 13.7*, *Section 13.13*, *Section 16.3*	0F84 TIBETAN MARK HALANTA 103A MYANMAR SIGN ASAT 193B LIMBU SIGN SA-I ABED MEETEI MAYEK APUN IYEK 11134 CHAKMA MAAYYAA
Ideographic-related		
Ideographic variation indication	*Section 6.2*	303E IDEOGRAPHIC VARIATION INDICATOR
Ideographic description	*Section 18.2*	2FF0..2FFB IDEOGRAPHIC DESCRIPTION CHARACTER LEFT TO RIGHT..IDEOGRAPHIC DESCRIPTION CHARACTER OVERLAID
Complex expression format control (scoped)		
Bidirectional ordering	*Section 23.2*	See *Table 23-3* for a full listing.

Mathematical expression processing and formatting	*Section 22.6*	2061 FUNCTION APPLICATION 2062 INVISIBLE TIMES 2063 INVISIBLE SEPARATOR 2064 INVISIBLE PLUS
Musical format control	*Section 21.2*	1D173 MUSICAL SYMBOL BEGIN BEAM 1D174 MUSICAL SYMBOL END BEAM 1D175 MUSICAL SYMBOL BEGIN TIE 1D176 MUSICAL SYMBOL END TIE 1D177 MUSICAL SYMBOL BEGIN SLUR 1D178 MUSICAL SYMBOL END SLUR 1D179 MUSICAL SYMBOL BEGIN PHRASE 1D17A MUSICAL SYMBOL END PHRASE
Prepended concatenation mark	*Section 9.2,* *Section 9.3,* *Section 15.2*	0600 ARABIC NUMBER SIGN 0601 ARABIC SIGN SANAH 0602 ARABIC FOOTNOTE MARKER 0603 ARABIC SIGN SAFHA 0604 ARABIC SIGN SAMVAT 0605 ARABIC NUMBER MARK ABOVE 06DD ARABIC END OF AYAH 070F SYRIAC ABBREVIATION MARK 0890 ARABIC POUND MARK ABOVE 0891 ARABIC PIASTRE MARK ABOVE 08E2 ARABIC DISPUTED END OF AYAH 110BD KAITHI NUMBER SIGN 110CD KAITHI NUMBER SIGN ABOVE
Interlinear annotation	*Section 23.8*	FFF9 INTERLINEAR ANNOTATION ANCHOR FFFA INTERLINEAR ANNOTATION SEPARATOR FFFB INTERLINEAR ANNOTATION TERMINATOR
Deprecated alternate formatting	*Section 23.3*	206A INHIBIT SYMMETRIC SWAPPING 206B ACTIVATE SYMMETRIC SWAPPING 206C INHIBIT ARABIC FORM SHAPING 206D ACTIVATE ARABIC FORM SHAPING 206E NATIONAL DIGIT SHAPES 206F NOMINAL DIGIT SHAPES
Other unusual format control		
Miao tonal vowel position control	*Section 18.10*	16F8F MIAO TONE RIGHT 16F90 MIAO TONE TOP RIGHT 16F91 MIAO TONE ABOVE 16F92 MIAO TONE BELOW
Shorthand format control	*Section 21.6*	1BC9D DUPLOYAN THICK LETTER SELECTOR 1BCA0 SHORTHAND FORMAT LETTER OVERLAP 1BCA1 SHORTHAND FORMAT CONTINUING OVERLAP 1BCA2 SHORTHAND FORMAT DOWN STEP 1BCA3 SHORTHAND FORMAT UP STEP
SignWriting fill and rotation	*Section 21.7*	1DA9B..1DA9F SIGNWRITING FILL MODIFIER-2..SIGNWRITING FILL MODIFIER-6 1DAA1..1DAAF SIGNWRITING ROTATION MODIFIER-2..SIGNWRITING ROTATION MODIFIER-16
Mongolian vowel separation	*Section 13.5*	180E MONGOLIAN VOWEL SEPARATOR
Variation selection		

Generic variation selectors	*Section 23.4*	FE00..FE0F VARIATION SELECTOR-1..VARIATION SELECTOR-16 E0100..E01EF VARIATION SELECTOR-17..VARIATION SELECTOR-256
Mongolian variation selectors	*Section 13.5*	180B MONGOLIAN FREE VARIATION SELECTOR ONE 180C MONGOLIAN FREE VARIATION SELECTOR TWO 180D MONGOLIAN FREE VARIATION SELECTOR THREE 180F MONGOLIAN FREE VARIATION SELECTOR FOUR
Emoji modifiers for skin tone	*Section 22.9*, UTS #51	1F3FB..1F3FF EMOJI MODIFIER FITZPATRICK TYPE-1-2..EMOJI MODIFIER FITZPATRICK TYPE-6
Emoji components to indicate hair style	UTS #51	1F9B0..1F9B3 EMOJI COMPONENT RED HAIR..EMOJI COMPONENT WHITE HAIR
Tag characters		
Deprecated language tag	*Section 23.9*	E0001 LANGUAGE TAG
Tag characters	*Section 23.9*	E0020..E007F TAG SPACE..CANCEL TAG
Miscellaneous		
Collation weighting and sequence interpretation	*Section 23.2*	034F COMBINING GRAPHEME JOINER
Byte order signature	*Section 23.8*	FEFF ZERO WIDTH NO-BREAK SPACE
Object replacement	*Section 23.8*	FFFC OBJECT REPLACEMENT CHARACTER
Code conversion fallback	*Section 23.8*	FFFD REPLACEMENT CHARACTER

4.13 Characters and Sequences That Should Not Be Emitted

There are certain text elements (see *Section 2.1.2, Text Elements, Characters, and Text Processes*) where more than one choice of representation by a character or character sequence would result in the same appearance. Many such cases are handled by the Unicode Normalization Algorithm, which rigorously defines canonical and compatibility equivalences, and which does not specify a preference for one normalization form over another. (See *Section 2.12, Equivalent Sequences*.) However, there are a smaller number of cases, some of which are salient to users, where an ambiguous representation may exist in the encoding, and where the Unicode Normalization Algorithm does not specify an equivalence. In such cases, the Unicode Standard instead often specifies recommended representations or spellings, to help in the interchange of text.

The data file DoNotEmit.txt in the Unicode Character Database lists such ambiguous characters or sequences together with their recommended alternative representation in machine-readable form. In newly authored text, these recommended sequences should be used and the non-recommended sequences should not be generated or emitted by implementations. The non-recommended alternatives are not *invalid* nor should they be flagged as errors if encountered in existing data. Implementations should continue to interpret and display any such non-recommended alternatives.

Applications such as input methods or auto corrections could be used to steer the user to the recommended alternative. Other implementations may use the information in the DoNotEmit.txt file to treat the listed characters or sequences as similar or identical to their recommended alternative for display, collation or searching purposes, or to suppress the non-recommended alternatives for identifiers.

The DoNotEmit.txt data file does not define any formal character properties. The information is intended to help with ambiguities in coded representation that cannot be resolved by applying the Unicode Normalization Algorithm; DoNotEmit.txt therefore omits listing of canonical equivalences. The listings in the file are neither comprehensive nor subject to a stability policy. Entries may be added or removed in subsequent versions of the Unicode Standard to reflect new information on usage.

Chapter 5

Implementation Guidelines

It is possible to implement a substantial subset of the Unicode Standard as "wide ASCII" with little change to existing programming practice. However, the Unicode Standard also provides for languages and writing systems that have more complex behavior than English does. Whether one is implementing a new operating system from the ground up or enhancing existing programming environments or applications, it is necessary to examine many aspects of current programming practice and conventions to deal with this more complex behavior.

This chapter covers a series of short, self-contained topics that are useful for implementers. The information and examples presented here are meant to help implementers understand and apply the design and features of the Unicode Standard. That is, they are meant to promote good practice in implementations conforming to the Unicode Standard.

These recommended guidelines are not normative and are not binding on the implementer, but are intended to represent best practice. When implementing the Unicode Standard, it is important to look not only at the letter of the conformance rules, but also at their spirit. Many of the following guidelines have been created specifically to assist people who run into issues with conformant implementations, while reflecting the requirements of actual usage.

5.1 Data Structures for Character Conversion

The Unicode Standard exists in a world of other text and character encoding standards—some private, some national, some international. A major strength of the Unicode Standard is the number of other important standards that it incorporates. In many cases, the Unicode Standard included duplicate characters to guarantee round-trip transcoding to established and widely used standards.

5.1.1 Issues

Conversion of characters between standards is not always a straightforward proposition. Many characters have mixed semantics in one standard and may correspond to more than one character in another. Sometimes standards give duplicate encodings for the same character; at other times the interpretation of a whole set of characters may depend on the application. Finally, there are subtle differences in what a standard may consider a character.

For these reasons, mapping tables are usually required to map between the Unicode Standard and another standard. Mapping tables need to be used consistently for text data exchange to avoid modification and loss of text data. For details, see Unicode Technical Standard #22, "Character Mapping Markup Language (CharMapML)." By contrast, conversions between different Unicode encoding forms are fast, lossless permutations.

There are important security issues associated with encoding conversion. For more information, see Unicode Technical Report #36, "Unicode Security Considerations."

The Unicode Standard can be used as a pivot to transcode among n different standards. This process, which is sometimes called *triangulation*, reduces the number of mapping tables that an implementation needs from $O(n^2)$ to $O(n)$.

5.1.2 Multistage Tables

Tables require space. Even small character sets often map to characters from several different blocks in the Unicode Standard and thus may contain up to 64K entries (for the BMP) or 1,088K entries (for the entire codespace) in at least one direction. Several techniques exist to reduce the memory space requirements for mapping tables. These techniques apply not only to transcoding tables, but also to many other tables needed to implement the Unicode Standard, including character property data, case mapping, collation tables, and glyph selection tables.

Flat Tables. If diskspace is not at issue, virtual memory architectures yield acceptable working set sizes even for flat tables because the frequency of usage among characters differs widely. Even small character sets contain many infrequently used

characters. In addition, data intended to be mapped into a given character set generally does not contain characters from all blocks of the Unicode Standard (usually, only a few blocks at a time need to be transcoded to a given character set). This situation leaves certain sections of the mapping tables unused—and therefore paged to disk. The effect is most pronounced for large tables mapping from the Unicode Standard to other character sets, which have large sections simply containing mappings to the default character, or the "unmappable character" entry.

Ranges. It may be tempting to "optimize" these tables for space by providing elaborate provisions for nested ranges or similar devices. This practice leads to unnecessary performance costs on modern, highly pipelined processor architectures because of branch penalties. A faster solution is to use an *optimized two-stage table*, which can be coded without any test or branch instructions. Hash tables can also be used for space optimization, although they are not as fast as multistage tables.

Two-Stage Tables. Two-stage tables are a commonly employed mechanism to reduce table size (see *Figure 5-1*). They use an array of pointers and a default value. If a pointer is NULL, the value returned by a lookup operation in the table is the default value. Otherwise, the pointer references a block of values used for the second stage of the lookup. For BMP characters, it is quite efficient to organize such two-stage tables in terms of high byte and low byte values. The first stage is an array of 256 pointers, and each of the secondary blocks contains 256 values indexed by the low byte in the code point. For supplementary characters, it is often advisable to structure the pointers and second-stage arrays somewhat differently, so as to take best advantage of the very sparse distribution of supplementary characters in the remaining codespace.

Figure 5-1. Two-Stage Tables
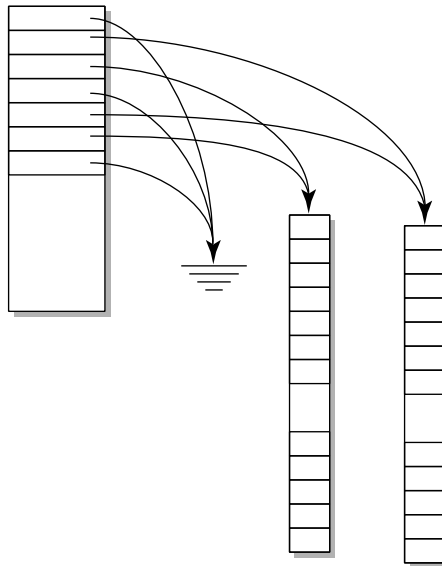
Optimized Two-Stage Table. Wherever any blocks are identical, the pointers just point to the same block. For transcoding tables, this case occurs generally for a block containing only mappings to the default or "unmappable" character. Instead of using NULL pointers and a default value, one "shared" block of default entries is created. This block is pointed to by all first-stage table entries, for which no character value can be mapped. By avoiding tests and branches, this strategy provides access time that approaches the simple array access, but at a great savings in storage.

Multistage Table Tuning. Given a table of arbitrary size and content, it is a relatively simple matter to write a small utility that can calculate the optimal number of stages and their width for a multistage table. Tuning the number of stages and the width of their arrays of index pointers can result in various trade-offs of table size versus average access time.

5.2 Programming Languages and Data Types

Programming languages provide for the representation and handling of characters and strings via data types, data constants (literals), and methods. Explicit support for Unicode helps with the development of multilingual applications. In some programming languages, strings are expressed as sequences (arrays) of primitive types, exactly corresponding to sequences of code units of one of the Unicode encoding forms. In other languages, strings are objects, but indexing into strings follows the semantics of addressing code units of a particular encoding form.

Data types for "characters" generally hold just a single Unicode code point value for low-level processing and lookup of character property values. When a primitive data type is used for single-code point values, a *signed* integer type can be useful; negative values can hold "sentinel" values like end-of-string or end-of-file, which can be easily distinguished from Unicode code point values. However, in most APIs, string types should be used to accommodate user-perceived characters, which may require sequences of code points.

5.2.1 Unicode Data Types for C

ISO/IEC Technical Report 19769, *Extensions for the programming language C to support new character types*, defines data types for the three Unicode encoding forms (UTF-8, UTF-16, and UTF-32), syntax for Unicode string and character literals, and methods for the conversion between the Unicode encoding forms. No other methods are specified.

Unicode strings are encoded as arrays of primitive types as usual. For UTF-8, UTF-16, and UTF-32, the basic types are char, char16_t, and char32_t, respectively. The ISO Technical Report assumes that char is at least 8 bits wide for use with UTF-8. While char and wchar_t may be signed or unsigned types, the new char16_t and char32_t types are defined to be unsigned integer types.

Unlike the specification in the wchar_t programming model, the Unicode data types do not require that a single string base unit alone (especially char or char16_t) must be able to store any one character (code point).

UTF-16 string and character literals are written with a lowercase u as a prefix, similar to the L prefix for wchar_t literals. UTF-32 literals are written with an uppercase U as a prefix. Characters outside the basic character set are available for use in string literals through the \uhhhh and \Uhhhhhhhh escape sequences.

These types and semantics are available in a compiler if the <uchar.h> header is present and defines the __STDC_UTF_16__ (for char16_t) and __STDC_UTF_32__ (for char32_t) macros.

Because Technical Report 19769 was not available when UTF-16 was first introduced, many implementations have been supporting a 16-bit `wchar_t` to contain UTF-16 code units. Such usage is not conformant to the C standard, because supplementary characters require use of pairs of `wchar_t` units in this case.

ANSI/ISO C wchar_t. With the `wchar_t` wide character type, ANSI/ISO C provides for inclusion of fixed-width, wide characters. ANSI/ISO C leaves the semantics of the wide character set to the specific implementation but requires that the characters from the portable C execution set correspond to their wide character equivalents by zero extension. The Unicode characters in the ASCII range U+0020 to U+007E satisfy these conditions. Thus, if an implementation uses ASCII to code the portable C execution set, the use of the Unicode character set for the `wchar_t` type, in either UTF-16 or UTF-32 form, fulfills the requirement.

The width of `wchar_t` is compiler-specific and can be as small as 8 bits. Consequently, programs that need to be portable across any C or C++ compiler should not use `wchar_t` for storing Unicode text. The `wchar_t` type is intended for storing compiler-defined wide characters, which may be Unicode characters in some compilers. However, programmers who want a UTF-16 implementation can use a macro or typedef (for example, `UNICHAR`) that can be compiled as `unsigned short` or `wchar_t` depending on the target compiler and platform. Other programmers who want a UTF-32 implementation can use a macro or typedef that might be compiled as `unsigned int` or `wchar_t`, depending on the target compiler and platform. This choice enables correct compilation on different platforms and compilers. Where a 16-bit implementation of `wchar_t` is guaranteed, such macros or typedefs may be predefined (for example, `TCHAR` on the Win32 API).

On systems where the native character type or `wchar_t` is implemented as a 32-bit quantity, an implementation may use the UTF-32 form to represent Unicode characters.

A limitation of the ISO/ANSI C model is its assumption that characters can always be processed in isolation. Implementations that choose to go beyond the ISO/ANSI C model may find it useful to mix widths within their APIs. For example, an implementation may have a 32-bit `wchar_t` and process strings in any of the UTF-8, UTF-16, or UTF-32 forms. Another implementation may have a 16-bit `wchar_t` and process strings as UTF-8 or UTF-16, but have additional APIs that process individual characters as UTF-32 or deal with pairs of UTF-16 code units.

5.3 Unknown and Missing Characters

This section briefly discusses how users or implementers might deal with characters that are not supported or that, although supported, are unavailable for legible rendering.

Reserved and Private-Use Character Codes. There are two classes of code points that even a "complete" implementation of the Unicode Standard cannot necessarily interpret correctly:

- Code points that are reserved
- Code points in the Private Use Area for which no private agreement exists

An implementation should not attempt to interpret such code points. However, in practice, applications must deal with unassigned code points or private-use characters. This may occur, for example, when the application is handling text that originated on a system implementing a later release of the Unicode Standard, with additional assigned characters.

Options for rendering such unknown code points include printing the code point as four to six hexadecimal digits, printing a black or white box, or another substitute glyph, such as that commonly shown for U+FFFD. For certain code points, it is common to display nothing; see "Default Ignorable Code Points" later in this section for details. An implementation should not blindly delete such characters, nor should it unintentionally transform them into something else.

Interpretable but Unrenderable Characters. An implementation may receive a code point to which a character is assigned in the Unicode Standard, but be unable to render it, because it lacks a font for the code point or is otherwise incapable of rendering it appropriately. In this case, the visual feedback an implementation is able to provide is limited. While it may have the resources to sort or line-break arbitrary text, any unrenderable character can only be indicated by some placeholder, such as displaying the hexadecimal value of the code point, or other meaningful information, such as the script of the character. For example, an unrenderable (but assigned) character can be displayed with distinctive glyphs that provide some meaningful indication of their type, such as Ⓐ, Ⓞ, and so on.

The Unicode Consortium maintains an open source Last Resort Font (https://github.com/unicode-org/last-resort-font/) that implementations can use for such fallback rendering of both assigned and unassigned code points.

Default Ignorable Code Points. Normally, characters outside the repertoire of supported characters for an implementation would be graphical characters displayed with a fallback glyph, such as a black box. However, certain special-use characters,

such as format controls or variation selectors, do not have visible glyphs of their own, although they may have an effect on the display of other characters. When such a special-use character is not supported by an implementation, it should not be displayed with a visible fallback glyph, but instead simply not be rendered at all. The list of such characters which should not be rendered with a fallback glyph is defined by the Default_Ignorable_Code_Point property in the Unicode Character Database. For more information, see *Section 5.21, Ignoring Characters in Processing*.

Interacting with Downlevel Systems. Versions of the Unicode Standard after Unicode 2.0 are strict supersets of Unicode 2.0 and all intervening versions. The Derived Age property tracks the version of the standard at which a particular character was added to the standard. This information can be particularly helpful in some interactions with downlevel systems. If the protocol used for communication between the systems provides for an announcement of the Unicode version on each one, an uplevel system can predict which recently added characters will appear as unassigned characters to the downlevel system.

5.4 Handling Surrogate Pairs in UTF-16

The method used by UTF-16 to address the 1,048,576 supplementary code points that cannot be represented by a single 16-bit value is called *surrogate pairs*. A surrogate pair consists of a high-surrogate code unit (leading surrogate) followed by a low-surrogate code unit (trailing surrogate), as described in the specifications in *Section 3.8, Surrogates*, and the UTF-16 portion of *Section 3.9, Unicode Encoding Forms*.

In well-formed UTF-16, a trailing surrogate can be preceded only by a leading surrogate and not by another trailing surrogate, a non-surrogate, or the start of text. A leading surrogate can be followed only by a trailing surrogate and not by another leading surrogate, a non-surrogate, or the end of text. Maintaining the well-formedness of a UTF-16 code sequence or accessing characters within a UTF-16 code sequence therefore puts additional requirements on some text processes. Surrogate pairs are designed to minimize this impact.

Leading surrogates and trailing surrogates are assigned to disjoint ranges of code units. In UTF-16, non-surrogate code points can never be represented with code unit values in those ranges. Because the ranges are disjoint, each code unit in well-formed UTF-16 must be one of the following:

- A single non-surrogate code unit, representing a code point between 0 and $D7FF_{16}$ or between $E000_{16}$ and $FFFF_{16}$
- A leading surrogate, representing the first part of a surrogate pair
- A trailing surrogate, representing the second part of a surrogate pair

By accessing at most two code units, a process using the UTF-16 encoding form can therefore interpret any Unicode character. Determining character boundaries requires at most scanning one preceding or one following code unit without regard to any other context.

As long as an implementation does not remove either of a pair of surrogate code units or incorrectly insert another character between them, the integrity of the data is maintained. Moreover, even if the data becomes corrupted, the corruption remains localized, unlike with some other multibyte encodings such as Shift-JIS or EUC. Corrupting a single UTF-16 code unit affects only a single character. Because of non-overlap (see *Section 2.5, Encoding Forms*), this kind of error does not propagate throughout the rest of the text.

UTF-16 enjoys a beneficial frequency distribution in that, for the majority of all text data, surrogate pairs will be very rare; non-surrogate code points, by contrast, will be very common. Not only does this help to limit the performance penalty incurred

when handling a variable-width encoding, but it also allows many processes either to take no specific action for surrogates or to handle surrogate pairs with existing mechanisms that are already needed to handle character sequences.

Implementations should fully support surrogate pairs in processing UTF-16 text. Without surrogate support, an implementation would not interpret any supplementary characters or guarantee the integrity of surrogate pairs. This might apply, for example, to an older implementation, conformant to Unicode Version 1.1 or earlier, before UTF-16 was defined. Support for supplementary characters is important because a significant number of them are relevant for modern use, despite their low frequency.

The individual *components* of implementations may have different levels of support for surrogates, as long as those components are assembled and communicate correctly. Low-level string processing, where a Unicode string is not interpreted but is handled simply as an array of code units, may ignore surrogate pairs. With such strings, for example, a truncation operation with an arbitrary offset might break a surrogate pair. (For further discussion, see *Section 2.7, Unicode Strings*.) For performance in string operations, such behavior is reasonable at a low level, but it requires higher-level processes to ensure that offsets are on character boundaries so as to guarantee the integrity of surrogate pairs.

Strategies for Surrogate Pair Support. Many implementations that handle advanced features of the Unicode Standard can easily be modified to support surrogate pairs in UTF-16. For example:

- Text collation can be handled by treating those surrogate pairs as "grouped characters," such as is done for "ij" in Dutch or "ch" in Slovak.

- Text entry can be handled by having a keyboard generate two Unicode code points with a single keypress, much as an ENTER key can generate CRLF or an Arabic keyboard can have a "*lam-alef*" key that generates a sequence of two characters, *lam* and *alef*.

- Truncation can be handled with the same mechanism as used to keep combining marks with base characters. For more information, see Unicode Standard Annex #29, "Unicode Text Segmentation."

Users are prevented from damaging the text if a text editor keeps *insertion points* (also known as *carets*) on character boundaries.

Implementations using UTF-8 and Unicode 8-bit strings necessitate similar considerations. The main difference from handling UTF-16 is that in the UTF-8 case the only characters that are represented with single code units (single bytes) in UTF-8 are the ASCII characters, U+0000..U+007F. Characters represented with multibyte sequences are very common in UTF-8, unlike surrogate pairs in UTF-16, which are

rather uncommon. This difference in frequency may result in different strategies for handling the multibyte sequences in UTF-8.

5.5 Handling Numbers

There are many sets of characters that represent decimal digits in different scripts. Systems that interpret those characters numerically should provide the correct numerical values. For example, the sequence <U+0968 DEVANAGARI DIGIT TWO, U+0966 DEVANAGARI DIGIT ZERO> when numerically interpreted has the value *twenty*.

When converting binary numerical values to a visual form, digits can be chosen from different scripts. For example, the value *twenty* can be represented either by <U+0032 DIGIT TWO, U+0030 DIGIT ZERO> or by <U+0968 DEVANAGARI DIGIT TWO, U+0966 DEVANAGARI DIGIT ZERO> or by <U+0662 ARABIC-INDIC DIGIT TWO, U+0660 ARABIC-INDIC DIGIT ZERO>. It is recommended that systems allow users to choose the format of the resulting digits by replacing the appropriate occurrence of U+0030 DIGIT ZERO with U+0660 ARABIC-INDIC DIGIT ZERO, and so on. (See *Chapter 4, Character Properties,* for the information needed to implement formatting and scanning numerical values.)

Fullwidth variants of the ASCII digits are simply compatibility variants of regular digits and should be treated as regular Western digits.

The Roman numerals, Greek acrophonic numerals, and East Asian ideographic numerals are decimal numeral writing systems, but they are not formally decimal radix digit systems. That is, it is not possible to do a one-to-one transcoding to forms such as 123456.789. Such systems are appropriate only for positive integer writing.

It is also possible to write numbers in two ways with CJK ideographic digits. For example, *Figure 22-6* shows how the number 1,234 can be written. Supporting these ideographic digits for numerical parsing means that implementations must be smart about distinguishing between the two cases.

Digits often occur in situations where they need to be parsed, but are not part of numbers. One such example is alphanumeric identifiers (see Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax").

Only in higher-level protocols, such as when implementing a full mathematical formula parser, do considerations such as superscripting and subscripting of digits become crucial for numerical interpretation.

See *Section 22.3, Numerals*, for a more extended discussion of the various types of numerals encoded in the Unicode Standard and their implications for implementations.

5.6 Normalization

Alternative Spellings. The Unicode Standard contains explicit codes for the most frequently used accented characters. These characters can also be composed; in the case of accented letters, characters can be composed from a base character and nonspacing mark(s).

The Unicode Standard provides decompositions for characters that can be composed using a base character plus one or more nonspacing marks. The decomposition mappings are specific to a particular version of the Unicode Standard. Further decomposition mappings may be added to the standard for new characters encoded in the future; however, no existing decomposition mapping for a currently encoded character will ever be removed or changed, nor will a decomposition mapping be added for a currently encoded character. These constraints on changes for decomposition are enforced by the Normalization Stability Policy. See the subsection "Policies" in *Section B.3, Other Unicode Online Resources*.

Normalization. Systems may normalize Unicode-encoded text to one particular sequence, such as normalizing composite character sequences into precomposed characters, or vice versa (see *Figure 5-2*).

Figure 5-2. Normalization
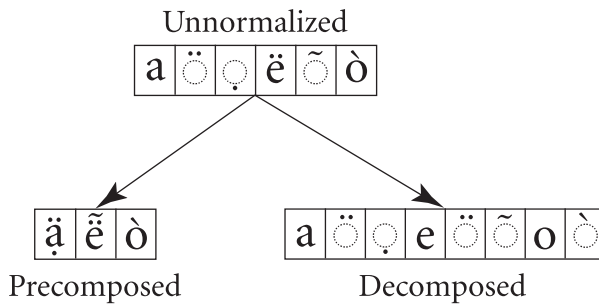
Compared to the number of *possible* combinations, only a relatively small number of precomposed base character plus nonspacing marks have independent Unicode character values.

Systems that cannot handle nonspacing marks can normalize to precomposed characters; this option can accommodate most modern Latin-based languages. Such systems can use fallback rendering techniques to at least visually indicate combinations that they cannot handle (see the "Fallback Rendering" subsection of *Section 5.13, Rendering Nonspacing Marks*).

In systems that *can* handle nonspacing marks, it may be useful to normalize so as to eliminate precomposed characters. This approach allows such systems to have a

homogeneous representation of composed characters and maintain a consistent treatment of such characters. However, in most cases, it does not require too much extra work to support mixed forms, which is the simpler route.

The Unicode Normalization Forms are defined in *Section 3.11, Normalization Forms*. For further information about implementation of normalization, see also Unicode Standard Annex #15, "Unicode Normalization Forms." For a general discussion of issues related to normalization, see "Equivalent Sequences" in *Section 2.2, Unicode Design Principles*; and *Section 2.11, Combining Characters*.

5.7 Compression

Using the Unicode character encoding may increase the amount of storage or memory space dedicated to the text portion of files. Compressing Unicode-encoded files or strings can therefore be an attractive option if the text portion is a large part of the volume of data compared to binary and numeric data, and if the processing overhead of the compression and decompression is acceptable.

Compression always constitutes a higher-level protocol and makes interchange dependent on knowledge of the compression method employed. For a detailed discussion of compression and a standard compression scheme for Unicode, see Unicode Technical Standard #6, "A Standard Compression Scheme for Unicode."

Encoding forms defined in *Section 2.5, Encoding Forms*, have different storage characteristics. For example, as long as text contains only characters from the Basic Latin (ASCII) block, it occupies the same amount of space whether it is encoded with the UTF-8 or ASCII codes. Conversely, text consisting of CJK ideographs encoded with UTF-8 will require more space than equivalent text encoded with UTF-16.

For processing rather than storage, the Unicode encoding form is usually selected for easy interoperability with existing APIs. Where there is a choice, the trade-off between decoding complexity (high for UTF-8, low for UTF-16, trivial for UTF-32) and memory and cache bandwidth (high for UTF-32, low for UTF-8 or UTF-16) should be considered.

5.8 Newline Guidelines

Newlines are represented on different platforms by carriage return (CR), line feed (LF), CRLF, or next line (NEL). Not only are newlines represented by different characters on different platforms, but they also have ambiguous behavior even on the same platform. These characters are often transcoded directly into the corresponding Unicode code points when a character set is transcoded; this means that even programs handling pure Unicode have to deal with the problems. Especially with the advent of the Web, where text on a single machine can arise from many sources, this causes a significant problem.

Newline characters are used to explicitly indicate line boundaries. For more information, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm." Newlines are also handled specially in the context of regular expressions. For information, see Unicode Technical Standard #18, "Unicode Regular Expressions." For the use of these characters in markup languages, see the W3C specification, "Unicode in XML and Other Markup Languages."

5.8.1 Definitions

Table 5-1 provides hexadecimal values for the acronyms used in these guidelines. The acronyms shown in *Table 5-1* correspond to characters or sequences of characters. The name column shows the usual names used to refer to the characters in question, whereas the other columns show the Unicode, ASCII, and EBCDIC encoded values for the characters.

Table 5-1. Hex Values for Acronyms

Acronym	Name	Unicode	ASCII	EBCDIC	
				Default	z/OS
CR	carriage return	000D	0D	0D	0D
LF	line feed	000A	0A	25	15
CRLF	carriage return and line feed	<000D 000A>	<0D 0A>	<0D 25>	<0D 15>
NEL	next line	0085	85	15	25
VT	vertical tab	000B	0B	0B	0B
FF	form feed	000C	0C	0C	0C
LS	line separator	2028	n/a	n/a	n/a
PS	paragraph separator	2029	n/a	n/a	n/a

Encoding. Except for LS and PS, the newline characters discussed here are encoded as control codes. Many control codes were originally designed for device control but, together with TAB, the newline characters are commonly used as part of plain text.

For more information on how Unicode encodes control codes, see *Section 23.1, Control Codes*.

Notation. This discussion of newline guidelines uses lowercase when referring to functions having to do with line determination, but uses the acronyms when referring to the actual characters involved. Keys on keyboards are indicated in all caps. For example:

> The line separator may be expressed by LS in Unicode text or CR on some platforms. It may be entered into text with the SHIFT-RETURN key.

EBCDIC. *Table 5-1* shows the two mappings of LF and NEL used by EBCDIC systems. The first EBCDIC column shows the default control code mapping of these characters, which is used in most EBCDIC environments. The second column shows the z/OS Unix System Services mapping of LF and NEL. That mapping arises from the use of the LF character for the newline function in C programs and in Unix environments, while text files on z/OS traditionally use NEL for the newline function.

NEL (next line) is not actually defined in 7-bit ASCII. It is defined in the ISO control function standard, ISO 6429, as a C1 control function. However, the 0x85 mapping shown in the ASCII column in *Table 5-1* is the usual way that this C1 control function is mapped in ASCII-based character encodings.

Newline Function. The acronym *NLF* (*newline function*) stands for the generic control function for indication of a new line break. It may be represented by different characters, depending on the platform, as shown in *Table 5-2*.

Table 5-2. NLF Platform Correlations

Platform	NLF Value
MacOS 9.x and earlier	CR
MacOS X	LF
Unix	LF
Windows	CRLF
EBCDIC-based OS	NEL

5.8.2 Line Separator and Paragraph Separator

A paragraph separator—independent of how it is encoded—is used to indicate a separation between paragraphs. A line separator indicates where a line break alone should occur, typically within a paragraph. For example:

> This is a paragraph with a line separator at this point,
> causing the word "causing" to appear on a different line, but not causing the

typical paragraph indentation, sentence breaking, line spacing, or change in flush (right, center, or left paragraphs).

For comparison, line separators basically correspond to HTML <br>, and paragraph separators to older usage of HTML <p> (modern HTML delimits paragraphs by enclosing them in <p>...</p>). In word processors, paragraph separators are usually entered using a keyboard RETURN or ENTER; line separators are usually entered using a modified RETURN or ENTER, such as SHIFT-ENTER.

A record separator is used to separate records. For example, when exchanging tabular data, a common format is to tab-separate the cells and use a CRLF at the end of a line of cells. This function is not precisely the same as line separation, but the same characters are often used.

Traditionally, *NLF* started out as a line separator (and sometimes record separator). It is still used as a line separator in simple text editors such as program editors. As platforms and programs started to handle word processing with automatic line-wrap, these characters were reinterpreted to stand for paragraph separators. For example, even such simple programs as the Windows Notepad program and the Mac Simple-Text program interpret their platform's *NLF* as a paragraph separator, not a line separator.

Once *NLF* was reinterpreted to stand for a paragraph separator, in some cases another control character was pressed into service as a line separator. For example, vertical tabulation VT is used in Microsoft Word. However, the choice of character for line separator is even less standardized than the choice of character for *NLF*.

Many Internet protocols and a lot of existing text treat *NLF* as a line separator, so an implementer cannot simply treat *NLF* as a paragraph separator in all circumstances.

5.8.3 Recommendations

The Unicode Standard defines two unambiguous separator characters: U+2029 PARAGRAPH SEPARATOR (PS) and U+2028 LINE SEPARATOR (LS). In Unicode text, the PS and LS characters should be used wherever the desired function is unambiguous. Otherwise, the following recommendations specify how to cope with an *NLF* when converting from other character sets to Unicode, when interpreting characters in text, and when converting from Unicode to other character sets.

Note that even if an implementer knows which characters represent *NLF* on a particular platform, CR, LF, CRLF, and NEL should be treated the same on input and in interpretation. Only on output is it necessary to distinguish between them.

Converting from Other Character Code Sets.

    R1 *If the exact usage of any NLF is known, convert it to LS or PS.*

> **R1a   If the exact usage of any NLF is unknown, remap it to the platform NLF.**

Recommendation R1a does not really help in interpreting Unicode text unless the implementer is the *only* source of that text, because another implementer may have left in LF, CR, CRLF, or NEL.

Interpreting Characters in Text.

> **R2   Always interpret PS as paragraph separator and LS as line separator.**

> **R2a   In word processing, interpret any NLF the same as PS.**

> **R2b   In simple text editors, interpret any NLF the same as LS.**

In line breaking, both PS and LS terminate a line; therefore, the Unicode Line Breaking Algorithm in Unicode Standard Annex #14, "Unicode Line Breaking Algorithm," is defined such that any NLF causes a line break.

> **R2c   In parsing, choose the safest interpretation.**

For example, in recommendation R2c an implementer dealing with sentence break heuristics would reason in the following way that it is safer to interpret any *NLF* as LS:

- Suppose an *NLF* were interpreted as LS, when it was meant to be PS. Because most paragraphs are terminated with punctuation anyway, this would cause misidentification of sentence boundaries in only a few cases.
- Suppose an *NLF* were interpreted as PS, when it was meant to be LS. In this case, line breaks would cause sentence breaks, which would result in significant problems with the sentence break heuristics.

Converting to Other Character Code Sets.

> **R3   If the intended target is known, map NLF, LS, and PS depending on the target conventions.**

For example, when mapping to Microsoft Word's internal conventions for documents, LS would be mapped to VT, and PS and any *NLF* would be mapped to CRLF.

> **R3a   If the intended target is unknown, map NLF, LS, and PS to the platform newline convention (CR, LF, CRLF, or NEL).**

In Java, for example, this is done by mapping to a string nlf, defined as follows:

```
String nlf = System.getProperty("line.separator");
```

Input and Output.

R4 **A `readline` function should stop at NLF, LS, FF, or PS. In the typical implementation, it does not include the NLF, LS, PS, or FF that caused it to stop.**

Because the separator is lost, the use of such a `readline` function is limited to text processing, where there is no difference among the types of separators.

R4a **A `writeline` (or `newline`) function should convert NLF, LS, and PS according to the recommendations R3 and R3a.**

In C, `gets` is defined to terminate at a newline and replaces the newline with `'\0'`, while `fgets` is defined to terminate at a newline and includes the newline in the array into which it copies the data. C implementations interpret `'\n'` either as LF or as the underlying platform newline *NLF,* depending on where it occurs. EBCDIC C compilers substitute the relevant codes, based on the EBCDIC execution set.

Page Separator. FF is commonly used as a page separator, and it should be interpreted that way in text. When displaying on the screen, it causes the text after the separator to be forced to the next page. It is interpreted in the same way as the LS for line breaking, in parsing, or in input segmentation such as `readline`. FF does not interrupt a paragraph, as paragraphs can and do span page boundaries.

5.9 Regular Expressions

Byte-oriented regular expression engines require extensions to handle Unicode successfully. The following issues are involved in such extensions:

- Unicode is a large character set—regular expression engines that are adapted to handle only small character sets may not scale well.

- Unicode encompasses a wide variety of languages that can have very different characteristics than English or other Western European text.

For detailed information on the requirements of Unicode regular expressions, see Unicode Technical Standard #18, "Unicode Regular Expressions."

5.10 Language Information in Plain Text

5.10.1 Requirements for Language Tagging

The requirement for language information embedded in plain text data is often over-stated. Many commonplace operations such as collation seldom require this extra information. In collation, for example, foreign language text is generally collated as if it were *not* in a foreign language. (See Unicode Technical Standard #10, "Unicode Collation Algorithm," for more information.) For example, an index in an English book would not sort the Slovak word "chlieb" after "czar," where it would be collated in Slovak, nor would an English atlas put the Swedish city of Örebro after Zanzibar, where it would appear in Swedish.

Text to speech is also an area where the case for embedded language information is overstated. Although language information may be useful in performing text-to-speech operations, modern software for doing acceptable text-to-speech must be so sophisticated in performing grammatical analysis of text that the extra work in determining the language is not significant in practice.

Language information can be useful in certain operations, such as spell-checking or hyphenating a mixed-language document. It is also useful in choosing the default font for a run of unstyled text; for example, the ellipsis character may have a very different appearance in Japanese fonts than in European fonts. Modern font and layout technologies produce different results based on language information. For example, the angle of the acute accent may be different for French and Polish.

5.10.2 Language Tags and Han Unification

A common misunderstanding about Unicode Han unification is the mistaken belief that Han characters cannot be rendered properly without language information. This idea might lead an implementer to conclude that language information must always be added to plain text using the tags. However, this implication is incorrect. The goal and methods of Han unification were to ensure that the text remained legible. Although font, size, width, and other format specifications need to be added to produce precisely the same appearance on the source and target machines, plain text remains legible in the absence of these specifications.

There should never be any confusion in Unicode, because the distinctions between the unified characters are all within the range of stylistic variations that exist in each country. No unification in Unicode should make it impossible for a reader to identify a character if it appears in a different font. Where precise font information is important, it is best conveyed in a rich text format.

Typical Scenarios. The following e-mail scenarios illustrate that the need for language information with Han characters is often overstated:

- Scenario 1. A Japanese user sends out untagged Japanese text. Readers are Japanese (with Japanese fonts). Readers see no differences from what they expect.

- Scenario 2. A Japanese user sends out an untagged mixture of Japanese and Chinese text. Readers are Japanese (with Japanese fonts) and Chinese (with Chinese fonts). Readers see the mixed text with only one font, but the text is still legible. Readers recognize the difference between the languages by the content.

- Scenario 3. A Japanese user sends out a mixture of Japanese and Chinese text. Text is marked with font, size, width, and so on, because the exact format is important. Readers have the fonts and other display support. Readers see the mixed text with different fonts for different languages. They recognize the difference between the languages by the content, and see the text with glyphs that are more typical for the particular language.

It is common even in printed matter to render passages of foreign language text in native- language fonts, just for familiarity. For example, Chinese text in a Japanese document is commonly rendered in a Japanese font.

5.11 Editing and Selection

As far as a user is concerned, the underlying representation of text is not a material concern, but it is important that an editing interface present a uniform implementation of what the user thinks of as characters. (See "'Characters' and Grapheme Clusters" in _Section 2.11, Combining Characters_.) The user expects them to behave as units in terms of mouse selection, arrow key movement, backspacing, and so on. For example, when such behavior is implemented, and an accented letter is represented by a sequence of base character plus a nonspacing combining mark, using the right arrow key would logically skip from the start of the base character to the end of the last nonspacing character.

In some cases, editing a user-perceived "character" or visual cluster element by element may be the preferred way. For example, a system might have the _backspace_ key delete by using the underlying code point, while the _delete_ key could delete an entire cluster. Moreover, because of the way keyboards and input method editors are implemented, there often may not be a one-to-one relationship between what the user thinks of as a character and the key or key sequence used to input it.

Three types of boundaries are generally useful in editing and selecting within words: cluster boundaries, stacked boundaries and atomic character boundaries.

**Cluster Boundaries.** Arbitrarily defined cluster boundaries may occur in scripts such as Devanagari, for which selection may be defined as applying to syllables or parts of syllables. In such cases, combining character sequences such as _ka + vowel sign a_ or conjunct clusters such as _ka + halant + ta_ are selected as a single unit. (See _Figure 5-3_.)

Figure 5-3. Consistent Character Boundaries
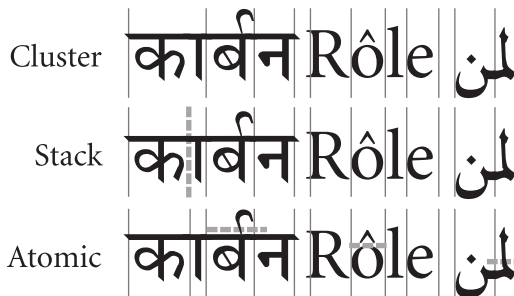
**Stacked Boundaries.** Stacked boundaries are generally somewhat finer than cluster boundaries. Free-standing elements (such as _vowel sign a_ in Devanagari) can be independently selected, but any elements that "stack" (including vertical ligatures such as Arabic _lam + meem_ in _Figure 5-3_) can be selected only as a single unit. Stacked

boundaries treat default grapheme clusters as single entities, much like composite characters. (See Unicode Standard Annex #29, "Unicode Text Segmentation," for the definition of default grapheme clusters and for a discussion of how grapheme clusters can be tailored to meet the needs of defining arbitrary cluster boundaries.)

Atomic Character Boundaries. The use of atomic character boundaries is closest to selection of individual Unicode characters. However, most modern systems indicate selection with some sort of rectangular highlighting. This approach places restrictions on the consistency of editing because some sequences of characters do not linearly progress from the start of the line. When characters stack, two mechanisms are used to visually indicate partial selection: linear and nonlinear boundaries.

Linear Boundaries. Use of linear boundaries treats the entire width of the resultant glyph as belonging to the first character of the sequence, and the remaining characters in the backing-store representation as having no width and being visually afterward.

This option is the simplest mechanism. The advantage of this system is that it requires very little additional implementation work. The disadvantage is that it is never easy to select narrow characters, let alone a zero-width character. Mechanically, it requires the user to select just to the right of the nonspacing mark and drag just to the left. It also does not allow the selection of individual nonspacing marks if more than one is present.

Nonlinear Boundaries. Use of nonlinear boundaries divides any stacked element into parts. For example, picking a point halfway across a *lam + meem* ligature can represent the division between the characters. One can either allow highlighting with multiple rectangles or use another method such as coloring the individual characters.

With more work, a precomposed character can behave in deletion as if it were a composed character sequence with atomic character boundaries. This procedure involves deriving the character's decomposition on the fly to get the components to be used in simulation. For example, deletion occurs by decomposing, removing the last character, then recomposing (if more than one character remains). However, this technique does not work in general editing and selection.

In most editing systems, the code point is the smallest addressable item, so the selection and assignment of properties (such as font, color, letterspacing, and so on) cannot be done on any finer basis than the code point. Thus the accent on an "e" could not be colored differently than the base in a precomposed character, although it could be colored differently if the text were stored internally in a decomposed form.

Just as there is no single notion of text element, so there is no single notion of editing character boundaries. At different times, users may want different degrees of granularity in the editing process. Two methods suggest themselves. First, the user may set a global preference for the character boundaries. Second, the user may have alterna-

tive command mechanisms, such as Shift-Delete, which give more (or less) fine control than the default mode.

5.12 Strategies for Handling Nonspacing Marks

By following these guidelines, a programmer should be able to implement systems and routines that provide for the effective and efficient use of nonspacing marks in a wide variety of applications and systems. The programmer also has the choice between minimal techniques that apply to the vast majority of existing systems and more sophisticated techniques that apply to more demanding situations, such as higher-end desktop publishing.

In this section and the following section, the terms *nonspacing mark* and *combining character* are used interchangeably. The terms *diacritic, accent, stress mark, Hebrew point, Arabic vowel*, and others are sometimes used instead of *nonspacing mark*. (They refer to particular types of nonspacing marks.) Properly speaking, a nonspacing mark is any combining character that does not add space along the writing direction. For a formal definition of nonspacing mark, see *Section 3.6, Combination*.

A relatively small number of implementation features are needed to support nonspacing marks. Different levels of implementation are also possible. A minimal system yields good results and is relatively simple to implement. Most of the features required by such a system are simply modifications of existing software.

As nonspacing marks are required for a number of writing systems, such as Arabic, Hebrew, and those of South Asia, many vendors already have systems capable of dealing with these characters and can use their experience to produce general-purpose software for handling these characters in the Unicode Standard.

Rendering. Composite character sequences can be rendered effectively by means of a fairly simple mechanism. In simple character rendering, a nonspacing combining mark has a zero advance width, and a composite character sequence will have the same width as the base character.

Wherever a sequence of base character plus one or more nonspacing marks occurs, the glyphs for the nonspacing marks can be positioned relative to the base. The ligature mechanisms in the fonts can also substitute a glyph representing the combined form. In some cases the width of the base should change because of an applied accent, such as with "î". The ligature or contextual form mechanisms in the font can be used to change the width of the base in cases where this is required.

Other Processes. Correct multilingual comparison routines must already be able to compare a sequence of characters as one character, or one character as if it were a sequence. Such routines can also handle combining character sequences when supplied with the appropriate data. When searching strings, remember to check for

additional nonspacing marks in the target string that may affect the interpretation of the last matching character.

Line breaking algorithms generally use state machines for determining word breaks. Such algorithms can be easily adapted to prevent separation of nonspacing marks from base characters. (See also the discussion in *Section 5.6, Normalization*. For details in particular contexts, see Unicode Technical Standard #10, "Unicode Collation Algorithm"; Unicode Standard Annex #14, "Unicode Line Breaking Algorithm"; and Unicode Standard Annex #29, "Unicode Text Segmentation.")

5.12.1 Keyboard Input

A common implementation for the input of combining character sequences is the use of *dead keys*. These keys match the mechanics used by typewriters to generate such sequences through overtyping the base character after the nonspacing mark. In computer implementations, keyboards enter a special state when a dead key is pressed for the accent and emit a precomposed character only when one of a limited number of "legal" base characters is entered. It is straightforward to adapt such a system to emit combining character sequences or precomposed characters as needed.

Typists, especially in the Latin script, are trained on systems that work using dead keys. However, many scripts in the Unicode Standard (including the Latin script) may be implemented according to the handwriting sequence, in which users type the base character first, *followed* by the accents or other nonspacing marks (see *Figure 5-4*).

Figure 5-4. Dead Keys Versus Handwriting Sequence
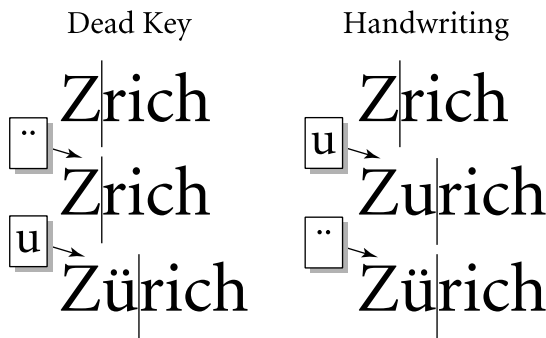
In the case of handwriting sequence, each keystroke produces a distinct, natural change on the screen; there are no hidden states. To add an accent to any existing character, the user positions the insertion point (*caret*) after the character and types the accent.

5.12.2 Truncation

There are two types of truncation: truncation by character count and truncation by displayed width. Truncation by character count can entail loss (be lossy) or be lossless.

Truncation by character count is used where, due to storage restrictions, a limited number of characters can be entered into a field; it is also used where text is broken into buffers for transmission and other purposes. The latter case can be lossless if buffers are recombined seamlessly before processing or if lookahead is performed for possible combining character sequences straddling buffers.

When fitting data into a field of limited storage length, some information will be lost. The preferred position for truncating text in that situation is on a grapheme cluster boundary. As *Figure 5-5* shows, such truncation can mean truncating at an earlier point than the last character that would have fit within the physical storage limitation. (See Unicode Standard Annex #29, "Unicode Text Segmentation.")

Figure 5-5. Truncating Grapheme Clusters

On Grapheme Cluster Boundaries

J o s e ◌́

Clipping

José

Ellipsis

Jo...
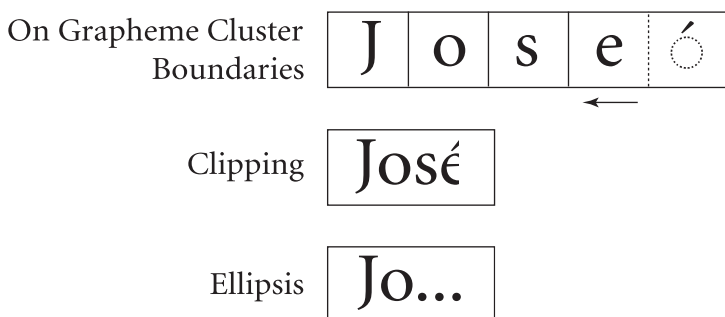
Truncation by displayed width is used for visual display in a narrow field. In this case, truncation occurs on the basis of the width of the resulting string rather than on the basis of a character count. In simple systems, it is easiest to truncate by width, starting from the end and working backward by subtracting character widths as one goes. Because a trailing nonspacing mark does not contribute to the measurement of the string, the result will not separate nonspacing marks from their base characters.

If the textual environment is more sophisticated, the widths of characters may depend on their context, due to effects such as kerning, ligatures, or contextual formation. For such systems, the width of a precomposed character, such as an "î", may be different than the width of a narrow base character alone. To handle these cases, a final check should be made on any truncation result derived from successive subtractions.

A different option is simply to clip the characters graphically. Unfortunately, this may result in clipping off part of a character, which can be visually confusing. Also, if the clipping occurs between characters, it may not give any visual feedback that characters are being omitted. A graphic or ellipsis can be used to give this visual feedback.

5.13 Rendering Nonspacing Marks

This discussion assumes the use of proportional fonts, where the widths of individual characters can vary. Various techniques can be used with monospaced fonts. In general, however, it is possible to get only a semblance of a correct rendering for most scripts in such fonts.

When rendering a sequence consisting of more than one nonspacing mark, the nonspacing marks should, by default, be stacked outward from the base character. That is, if two nonspacing marks appear over a base character, then the first nonspacing mark should appear on top of the base character, and the second nonspacing mark should appear on top of the first. If two nonspacing marks appear under a base character, then the first nonspacing mark should appear beneath the base character, and the second nonspacing mark should appear below the first (see *Section 2.11, Combining Characters*). This default treatment of multiple, potentially interacting nonspacing marks is known as the inside-out rule (see *Figure 5-6*).

Figure 5-6. Inside-Out Rule
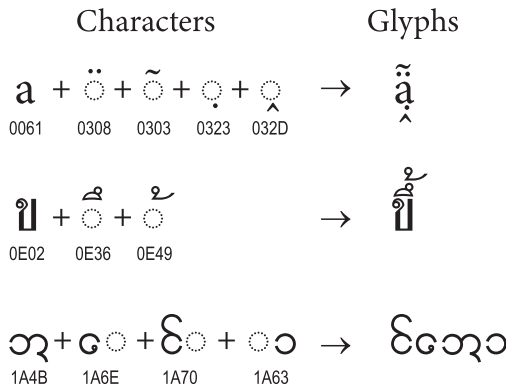
This default behavior may be altered based on typographic preferences or on knowledge of the specific orthographic treatment to be given to multiple nonspacing marks in the context of a particular writing system. For example, in the modern Vietnamese writing system, an acute or grave accent (serving as a tone mark) may be positioned slightly to one side of a circumflex accent rather than directly above it. If the text to be displayed is known to employ a different typographic convention (either implicitly through knowledge of the language of the text or explicitly through rich text-style bindings), then an alternative positioning may be given to multiple nonspacing marks instead of that specified by the default inside-out rule.

Fallback Rendering. Several methods are available to deal with an unknown composed character sequence that is outside of a fixed, renderable set (see *Figure 5-7*). One method (*Show Hidden*) indicates the inability to draw the sequence

by drawing the base character first and then rendering the nonspacing mark as an individual unit, with the nonspacing mark positioned on a dotted circle. (This convention is used in the Unicode code charts.)

Figure 5-7. Fallback Rendering

Ĝĝ G◌̂g◌̂ Ĝĝ
"Ideal"   "Show   "Simple
          Hidden" Overlap"

Another method (*Simple Overlap*) uses a default fixed position for an overlapping zero-width nonspacing mark. This position is generally high enough to make sure that the mark does not collide with capital letters. This will mean that this mark is placed too high above many lowercase letters. For example, the default positioning of a circumflex can be above the ascent, which will place it above capital letters. Even though the result will not be particularly attractive for letters such as *g-circumflex*, the result should generally be recognizable in the case of single nonspacing marks.

In a degenerate case, a nonspacing mark occurs as the first character in the text or is separated from its base character by a *line separator*, *paragraph separator*, or other format character that causes a positional separation. This result is called a defective combining character sequence (see *Section 3.6, Combination*). Defective combining character sequences should be rendered as if they had a *no-break space* as a base character. (See *Section 7.9, Combining Marks*.)

Bidirectional Positioning. In bidirectional text, the nonspacing marks are reordered *with* their base characters; that is, they visually apply to the same base character after the algorithm is used (see *Figure 5-8*). There are a few ways to accomplish this positioning.

Figure 5-8. Bidirectional Placement
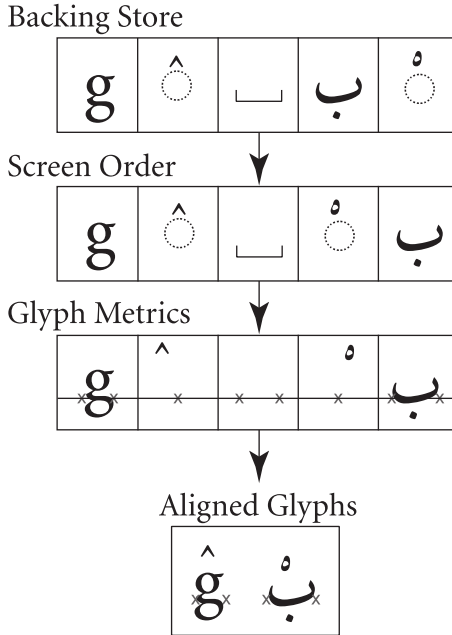
The simplest method is similar to the *Simple Overlap* fallback method. In the Bidirectional Algorithm, combining marks take the level of their base character. In that case, Arabic and Hebrew nonspacing marks would come to the left of their base characters. The font is designed so that instead of overlapping to the left, the Arabic and Hebrew nonspacing marks overlap to the right. In *Figure 5-8*, the "glyph metrics" line shows the pen start and end for each glyph with such a design. After aligning the start and end points, the final result shows each nonspacing mark attached to the corresponding base letter. More sophisticated rendering could then apply the positioning methods outlined in the next section.

Some rendering software may require keeping the nonspacing mark glyphs consistently ordered to the right of the base character glyphs. In that case, a second pass can be done after producing the "screen order" to put the odd-level nonspacing marks on the right of their base characters. As the levels of nonspacing marks will be the same as their base characters, this pass can swap the order of nonspacing mark glyphs and base character glyphs in right-to-left (odd) levels. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm.")

Justification. Typically, full justification of text adds extra space at space characters so as to widen a line; however, if there are too few (or no) space characters, some systems add extra letterspacing between characters (see *Figure 5-9*). This process needs to be modified if zero-width nonspacing marks are present in the text. Other-

wise, if extra justifying space is added after the base character, it can have the effect of visually separating the nonspacing mark from its base.

Figure 5-9. Justification

Zürich

Z u ̈ r i c h   66 points/6 positions
= 11 points per position

Z ü r i c h   66 points/5 positions
= 13.2 points per position
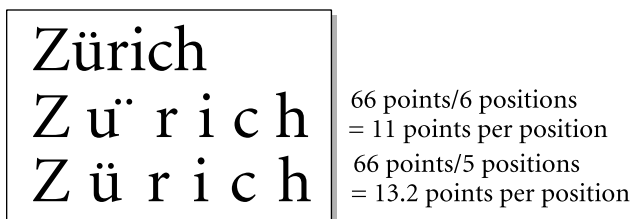
Because nonspacing marks always follow their base character, proper justification adds letterspacing between characters only if the second character is a base character.

5.13.1 Canonical Equivalence

Canonical equivalence must be taken into account in rendering multiple accents, so that any two canonically equivalent sequences display as the same. This is particularly important when the canonical order is not the customary keyboarding order, which happens in Arabic with vowel signs or in Hebrew with points. In those cases, a rendering system may be presented with either the typical typing order or the canonical order resulting from normalization, as shown in *Table 5-3*.

Table 5-3. Typing Order Differing from Canonical Order

Typical Typing Order	Canonical Order
U+0631 ر ARABIC LETTER REH + U+0651 ّ ARABIC SHADDA + U+064B ً ARABIC FATHATAN	U+0631 ر ARABIC LETTER REH + U+064B ً ARABIC FATHATAN + U+0651 ّ ARABIC SHADDA

With a restricted repertoire of nonspacing mark sequences, such as those required for Arabic, a ligature mechanism can be used to get the right appearance, as described earlier. When a fallback mechanism for placing accents based on their combining class is employed, the system should logically reorder the marks before applying the mechanism.

Rendering systems should handle *any* of the canonically equivalent orders of combining marks. This is not a performance issue: the amount of time necessary to reorder combining marks is insignificant compared to the time necessary to carry out other work required for rendering.

A rendering system can reorder the marks internally if necessary, as long as the resulting sequence is canonically equivalent. In particular, any permutation of the non-zero combining class values can be used for a canonical-equivalent internal

ordering. For example, a rendering system could internally permute weights to have U+0651 ARABIC SHADDA precede all vowel signs. This would use the remapping shown in *Table 5-4*.

Table 5-4. Permuting Combining Class Weights

Combining Class	Internal Weight
27	→ 33
28	→ 27
29	→ 28
30	→ 29
31	→ 30
32	→ 31
33	→ 32

Only non-zero combining class values can be changed, and they can be permuted *only*, not combined or split. This can be restated as follows:

- Two characters that have the same combining class values cannot be given distinct internal weights.

- Two characters that have distinct combining class values cannot be given the same internal weight.

- Characters with a combining class of zero must be given an internal weight of zero.

5.13.2 Positioning Methods

A number of methods are available to position nonspacing marks so that they are in the correct location relative to the base character and previous nonspacing marks.

Positioning with Ligatures. A fixed set of combining character sequences can be rendered effectively by means of fairly simple substitution, as shown in *Figure 5-10*.

Figure 5-10. Positioning with Ligatures

$$a \; + \; \ddot{\circ} \; \rightarrow \; \ddot{a}$$
$$A \; + \; \ddot{\circ} \; \rightarrow \; \ddot{A}$$
$$f \; + \; i \; \rightarrow \; fi$$

Wherever the glyphs representing a sequence of <base character, nonspacing mark> occur, a glyph representing the combined form is substituted. Because the

nonspacing mark has a zero advance width, the composed character sequence will automatically have the same width as the base character. More sophisticated text rendering systems may take additional measures to account for those cases where the composed character sequence kerns differently or has a slightly different advance width than the base character.

Positioning with ligatures is perhaps the simplest method of supporting nonspacing marks. Whenever there is a small, fixed set, such as those corresponding to the precomposed characters of ISO/IEC 8859-1 (Latin-1), this method is straightforward to apply. Because the composed character sequence almost always has the same width as the base character, rendering, measurement, and editing of these characters are much easier than for the general case of ligatures.

If a combining character sequence does not form a ligature, then either positioning with contextual forms or positioning with enhanced kerning can be applied. If they are not available, then a fallback method can be used.

Positioning with Contextual Forms. A more general method of dealing with positioning of nonspacing marks is to use contextual formation (see *Figure 5-11*). In this case for Devanagari, a consonant RA is rendered with a nonspacing glyph (*reph*) positioned above a base consonant. (See "Rendering Devanagari" in *Section 12.1, Devanagari.*) Depending on the position of the stem for the corresponding base consonant glyph, a contextual choice is made between *reph* glyphs with different side bearings, so that the tip of the *reph* will be placed correctly with respect to the base consonant's stem. Base glyphs generally fall into a fairly small number of classes, depending on their general shape and width, so a corresponding number of contextually distinct glyphs for the nonspacing mark suffice to produce correct rendering.

Figure 5-11. Positioning with Contextual Forms
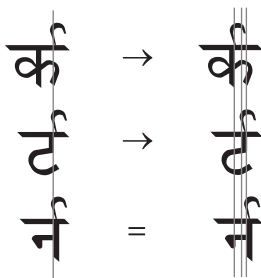
In general cases, a number of different heights of glyphs can be chosen to allow stacking of glyphs, at least for a few deep. (When these bounds are exceeded, then the fallback methods can be used.) This method can be combined with the ligature method so that in specific cases ligatures can be used to produce fine variations in position and shape.

Positioning with Enhanced Kerning. A third technique for positioning diacritics is an extension of the normal process of kerning to be both horizontal and vertical (see *Figure 5-12*). Typically, kerning maps from pairs of glyphs to a positioning offset. For example, in the word "To" the "o" should nest slightly under the "T". An extension of this system maps to both a *vertical* and a *horizontal* offset, allowing glyphs to be positioned arbitrarily.

Figure 5-12. Positioning with Enhanced Kerning
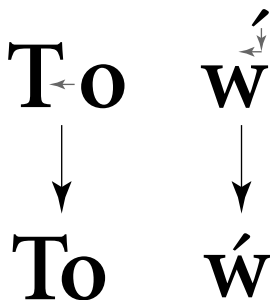
For effective use in the general case, the kerning process must be extended to handle more than simple kerning pairs, as multiple diacritics may occur after a base letter.

Positioning with enhanced kerning can be combined with the ligature method so that in specific cases ligatures can be used to produce fine variations in position and shape.

5.14 Locating Text Element Boundaries

A string of Unicode-encoded text often needs to be broken up into text elements programmatically. Common examples of text elements include what users think of as characters, words, lines, and sentences. The precise determination of text elements may vary according to locale, even as to what constitutes a "character." The goal of matching user perceptions cannot always be met, because the text alone does not always contain enough information to decide boundaries unambiguously. For example, the *period* (U+002E FULL STOP) is used ambiguously—sometimes for end-of-sentence purposes, sometimes for abbreviations, and sometimes for numbers. In most cases, however, programmatic text boundaries can match user perceptions quite closely, or at least not surprise the user.

Rather than concentrate on algorithmically searching for text elements themselves, a simpler computation looks instead at detecting the *boundaries* between those text elements. Precise definitions of the default Unicode mechanisms for determining such text element boundaries are found in Unicode Standard Annex #14, "Unicode Line Breaking Algorithm," and in Unicode Standard Annex #29, "Unicode Text Segmentation."

5.15 Identifiers

A common task facing an implementer of the Unicode Standard is the provision of a parsing and/or lexing engine for identifiers. To assist in the standard treatment of identifiers in Unicode character-based parsers, a set of guidelines is provided in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax," as a recommended default for the definition of identifier syntax. That document provides details regarding the syntax and conformance considerations. Associated data files defining the character properties referred to by the identifier syntax can be found in the Unicode Character Database.

5.16 Sorting and Searching

Sorting and searching overlap in that both implement degrees of *equivalence* of terms to be compared. In the case of searching, equivalence defines when terms match (for example, it determines when case distinctions are meaningful). In the case of sorting, equivalence affects the proximity of terms in a sorted list. These determinations of equivalence often depend on the application and language, but for an implementation supporting the Unicode Standard, sorting and searching must always take into account the Unicode character equivalence and canonical ordering defined in <u>Chapter 3, Conformance</u>.

5.16.1 Culturally Expected Sorting and Searching

Sort orders vary from culture to culture, and many specific applications require variations. Sort order can be by word or sentence, case-sensitive or case-insensitive, ignoring accents or not. It can also be either phonetic or based on the appearance of the character, such as ordering by stroke and radical for East Asian ideographs. Phonetic sorting of Han characters requires use of either a lookup dictionary of words or special programs to maintain an associated phonetic spelling for the words in the text.

Languages vary not only regarding which types of sorts to use (and in which order they are to be applied), but also in what constitutes a fundamental element for sorting. For example, Swedish treats U+00C4 LATIN CAPITAL LETTER A WITH DIAERESIS as an individual letter, sorting it after *z* in the alphabet; German, however, sorts it either like *ae* or like other accented forms of *ä* following *a*. Spanish traditionally sorted the digraph *ll* as if it were a letter between *l* and *m*. Examples from other languages (and scripts) abound.

As a result, it is not possible either to arrange characters in an encoding such that simple binary string comparison produces the desired collation order or to provide single-level sort-weight tables. The latter implies that character encoding details have only an indirect influence on culturally expected sorting.

Unicode Technical Standard #10, "Unicode Collation Algorithm" (UCA), describes the issues involved in culturally appropriate sorting and searching, and provides a specification for how to compare two Unicode strings while remaining conformant to the requirements of the Unicode Standard. The UCA also supplies the Default Unicode Collation Element Table as the data specifying the default collation order. Searching algorithms, whether brute-force or sublinear, can be adapted to provide language-sensitive searching as described in the UCA.

5.16.2 Language-Insensitive Sorting

In some circumstances, an application may need to do language-insensitive sorting—that is, sorting of textual data without consideration of language-specific cultural expectations about how strings should be ordered. For example, a temporary index may need only to be in *some* well-defined order, but the exact details of the order may not matter or be visible to users. However, even in these circumstances, implementers should be aware of some issues.

First, some subtle differences arise in binary ordering between the three Unicode encoding forms. Implementations that need to do only binary comparisons between Unicode strings still need to take this issue into account so as not to create interoperability problems between applications using different encoding forms. See *Section 5.17, Binary Order*, for further discussion.

Many applications of sorting or searching need to be case-insensitive, even while not caring about language-specific differences in ordering. This is the result of the design of protocols that may be very old but that are still of great current relevance. Traditionally, implementations did case-insensitive comparison by effectively mapping both strings to uppercase before doing a binary comparison. This approach is, however, not more generally extensible to the full repertoire of the Unicode Standard. The correct approach to case-insensitive comparison is to make use of case folding, as described in *Section 5.18, Case Mappings*.

5.16.3 Searching

Searching is subject to many of the same issues as comparison. Other features are often added, such as only matching words (that is, where a word boundary appears on each side of the match). One technique is to code a fast search for a weak match. When a candidate is found, additional tests can be made for other criteria (such as matching diacriticals, word match, case match, and so on).

When searching strings, it is necessary to check for trailing nonspacing marks in the target string that may affect the interpretation of the last matching character. That is, a search for "San Jose" may find a match in the string "Visiting San José, Costa Rica, is a...". If an exact (diacritic) match is desired, then this match should be rejected. If a weak match is sought, then the match should be accepted, but any trailing nonspacing marks should be included when returning the location and length of the target substring. The mechanisms discussed in Unicode Standard Annex #29, "Unicode Text Segmentation," can be used for this purpose.

One important application of weak equivalence is case-insensitive searching. Many traditional implementations map both the search string and the target text to uppercase. However, case mappings are language-dependent and *not* unambiguous. The

preferred method of implementing case insensitivity is described in <u>*Section 5.18, Case Mappings*</u>.

A related issue can arise because of inaccurate mappings from external character sets. To deal with this problem, characters that are easily confused by users can be kept in a weak equivalency class (đ *d-bar*, ð *eth*, Đ *capital d-bar*, Đ *capital eth*). This approach tends to do a better job of meeting users' expectations when searching for named files or other objects.

5.16.4 Sublinear Searching

International searching is clearly possible using the information in the collation, just by using brute force. However, this tactic requires an $O(m*n)$ algorithm in the worst case and an $O(m)$ algorithm in common cases, where n is the number of characters in the pattern that is being searched for and m is the number of characters in the target to be searched.

A number of algorithms allow for fast searching of simple text, using sublinear algorithms. These algorithms have only $O(m/n)$ complexity in common cases by skipping over characters in the target. Several implementers have adapted one of these algorithms to search text pre-transformed according to a collation algorithm, which allows for fast searching with native-language matching (see <u>*Figure 5-13*</u>).

Figure 5-13. Sublinear Searching

The main problems with adapting a language-aware collation algorithm for sublinear searching relate to multiple mappings and ignorables. Additionally, sublinear algorithms precompute tables of information. Mechanisms like the two-stage tables shown in <u>*Figure 5-1*</u> are efficient tools in reducing memory requirements.

5.17 Binary Order

When comparing text that is visible to end users, a correct linguistic sort should be used, as described in *Section 5.16, Sorting and Searching*. However, in many circumstances the only requirement is for a fast, well-defined ordering. In such cases, a binary ordering can be used.

Not all encoding forms of Unicode have the same binary order. UTF-8 and UTF-32 data, and UTF-16 data containing only BMP characters, sort in code point order, whereas UTF-16 data containing a mix of BMP and supplementary characters does not. This is because supplementary characters are encoded in UTF-16 with pairs of surrogate code units that have lower values ($D800_{16}$..$DFFF_{16}$) than some BMP code points.

Furthermore, when UTF-16 or UTF-32 data is serialized using one of the Unicode encoding schemes and compared byte-by-byte, the resulting byte sequences may or may not have the same binary ordering, because swapping the order of bytes will affect the overall ordering of the data. Due to these factors, text in the UTF-16BE, UTF-16LE, and UTF-32LE encoding schemes does not sort in code point order.

In general, the default binary sorting order for Unicode text should be code point order. However, it may be necessary to match the code unit ordering of a particular encoding form (or the byte ordering of a particular encoding scheme) so as to duplicate the ordering used in a different application.

Some sample routines are provided here for sorting one encoding form in the binary order of another encoding form.

5.17.1 UTF-8 in UTF-16 Order

The following comparison function for UTF-8 yields the same results as UTF-16 binary comparison. In the code, notice that it is necessary to do extra work only once per string, not once per byte. That work can consist of simply remapping through a small array; there are no extra conditional branches that could slow down the processing.

```
int strcmp8like16(unsigned char* a, unsigned char* b) {
  while (true) {
    int ac = *a++;
    int bc = *b++;
    if (ac != bc) return rotate[ac] - rotate[bc];
    if (ac == 0) return 0;
  }
}
```

```
static char rotate[256] =
      {0x00,  ...,                                        0x0F,
       0x10,  ...,                                        0x1F,
       .                                                  .
       .                                                  .
       .                                                  .
       0xD0,  ...,                                        0xDF,
       0xE0,  ...,                              0xED, 0xF3, 0xF4,
       0xEE, 0xEF, 0xF0, 0xF1, 0xF2, 0xF5,  ..., 0xFF};
```

The rotate array is formed by taking an array of 256 bytes from 0x00 to 0xFF, and rotating 0xEE to 0xF4, the initial byte values of UTF-8 for the code points in the range U+E000..U+10FFFF. These rotated values are shown in boldface. When this rotation is performed on the initial bytes of UTF-8, it has the effect of making code points U+10000..U+10FFFF sort below U+E000..U+FFFF, thus mimicking the ordering of UTF-16.

5.17.2 UTF-16 in UTF-8 Order

The following code can be used to sort UTF-16 in code point order. As in the routine for sorting UTF-8 in UTF-16 order, the extra cost is incurred once per function call, not once per character.

```
int strcmp16like8(Unichar* a, Unichar* b) {
  while (true) {
    int ac = *a++;
    int bc = *b++;
    if (ac != bc) {
      return (Unichar)(ac + utf16Fixup[ac>>11]) -
             (Unichar)(bc + utf16Fixup[bc>>11]);
    }
    if (ac == 0) return 0;
  }
}

static const Unichar utf16Fixup[32]={
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0x2000, 0xf800, 0xf800, 0xf800, 0xf800
};
```

This code uses Unichar as an unsigned 16-bit integral type. The construction of the utf16Fixup array is based on the following concept. The range of UTF-16 values is divided up into thirty-two 2K chunks. The 28th chunk corresponds to the values 0xD800..0xDFFF—that is, the surrogate code units. The 29th through 32nd chunks correspond to the values 0xE000..0xFFFF. The addition of 0x2000 to the surrogate

code units rotates them up to the range 0xF800..0xFFFF. Adding 0xF800 to the values 0xE000..0xFFFF and ignoring the unsigned integer overflow rotates them down to the range 0xD800..0xF7FF. Calculating the final difference for the return from the rotated values produces the same result as basing the comparison on code points, rather than the UTF-16 code units. The use of the hack of unsigned integer overflow on addition avoids the need for a conditional test to accomplish the rotation of values.

Note that this mechanism works correctly only on well-formed UTF-16 text. A modified algorithm must be used to operate on 16-bit Unicode strings that could contain isolated surrogates.

5.18 Case Mappings

Case is a normative property of characters in specific alphabets such as Latin, Greek, Cyrillic, Armenian, and archaic Georgian, whereby characters are considered to be variants of a single letter. These variants, which may differ markedly in shape and size, are called the uppercase letter (also known as capital or majuscule) and the lowercase letter (also known as small or minuscule). The uppercase letter is generally larger than the lowercase letter. Alphabets with case differences are called *bicameral*; those without are called *unicameral*.

The case mappings in the Unicode Character Database (UCD) are normative. This follows from their use in defining the case foldings in CaseFolding.txt and from the use of case foldings to define case-insensitive identifiers in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax." However, the normative status of case mappings does not preclude the adaptation of case mapping processes to local conventions, as discussed below. See also the Unicode Common Locale Data Repository (CLDR), https://cldr.unicode.org, for extensive data regarding local and language-specific casing conventions.

5.18.1 Titlecasing

Titlecasing refers to a casing practice wherein the first letter of a word is an uppercase letter and the rest of the letters are lowercase. This typically applies, for example, to initial words of sentences and to proper nouns. Depending on the language and orthographic practice, this convention may apply to other words as well, as for common nouns in German.

Titlecasing also applies to entire strings, as in instances of headings or titles of documents, for which multiple words are titlecased. The choice of which words to titlecase in headings and titles is dependent on language and local conventions. For example, "The Merry Wives of Windsor" is the appropriate titlecasing of that play's name in English, with the word "of" not titlecased. In German, however, the title is "Die lustigen Weiber von Windsor," and both "lustigen" and "von" are not titlecased. In French even fewer words are titlecased: "Les joyeuses commères de Windsor."

Moreover, the determination of what actually constitutes a word is language dependent, and this can influence which letter or letters of a "word" are uppercased when titlecasing strings. For example *l'arbre* is considered two words in French, whereas *can't* is considered one word in English.

The need for a normative Titlecase_Mapping property in the Unicode Standard derives from the fact that the standard contains certain digraph characters for compatibility. These digraph compatibility characters, such as U+01F3 "dz" LATIN SMALL LETTER DZ, require one form when being uppercased, U+01F1 "DZ" LATIN

CAPITAL LETTER DZ, and another form when being titlecased, U+01F2 "Dz" LATIN CAPITAL LETTER D WITH SMALL LETTER Z. The latter form is informally referred to as a *titlecase character*, because it is mixed case, with the first letter uppercase. Most characters in the standard have identical values for their Titlecase_Mapping and Uppercase_Mapping; however, the two values are distinguished for these few digraph compatibility characters.

5.18.2 Complications for Case Mapping

A number of complications to case mappings occur once the repertoire of characters is expanded beyond ASCII.

Change in Length. Case mappings may produce strings of different lengths than the original. For example, the German character U+00DF ß LATIN SMALL LETTER SHARP S expands when uppercased to the sequence of two characters "SS". Such expansion also occurs where there is no precomposed character corresponding to a case mapping, such as with U+0149 'n LATIN SMALL LETTER N PRECEDED BY APOS-TROPHE. The maximum string expansion as a result of case mapping in the Unicode Standard is three. For example, uppercasing U+0390 ΐ GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS results in three characters.

The lengths of case-mapped strings may also differ from their originals depending on the Unicode encoding form. For example, the Turkish strings "topkapı" (with a *dotless i*) and "TOPKAPI" have the same number of characters and are the same length in UTF-16 and UTF-32; however, in UTF-8, the representation of the upper-case form takes only seven bytes, whereas the lowercase form takes eight bytes. By comparison, the German strings "heiß" and "HEISS" have a different number of characters and differ in length in UTF-16 and UTF-32, but in UTF-8 both strings are encoded using the same number of bytes.

Greek iota subscript. The character U+0345 ͅ COMBINING GREEK YPOGEGRAM-MENI (*iota subscript*) requires special handling. As discussed in *Section 7.2, Greek*, the iota-subscript characters used to represent ancient text have special case mappings. Normally, the uppercase and lowercase forms of alpha-iota-subscript will map back and forth. In some instances, uppercase words should be transformed into their older spellings by removing accents and changing the iota subscript into a capital iota (and perhaps even removing spaces).

Context-dependent Case Mappings. Characters may have different case mappings, depending on the context surrounding the character in the original string. For example, U+03A3 "Σ" GREEK CAPITAL LETTER SIGMA lowercases to U+03C3 "σ" GREEK SMALL LETTER SIGMA if it is followed by another letter, but lowercases to U+03C2 "ς" GREEK SMALL LETTER FINAL SIGMA if it is not.

Because only a few context-sensitive case mappings exist, and because they involve only a very few characters, implementations may choose to hard-code the treatment of these characters for casing operations rather than using data-driven code based on the Unicode Character Database. However, if this approach is taken, each time the implementation is upgraded to a new version of the Unicode Standard, hard-coded casing operations should be checked for consistency with the updated data. See SpecialCasing.txt in the Unicode Character Database for details of context-sensitive case mappings.

Locale-dependent Case Mappings. The principal example of a case mapping that depends on the locale is Turkish, where U+0131 "ı" LATIN SMALL LETTER DOTLESS I maps to U+0049 "I" LATIN CAPITAL LETTER I and U+0069 "i" LATIN SMALL LETTER I maps to U+0130 "İ" LATIN CAPITAL LETTER I WITH DOT ABOVE. *Figure 5-14* shows the uppercase mapping for Turkish *i* and canonically equivalent sequences.

Figure 5-14. Uppercase Mapping for Turkish I
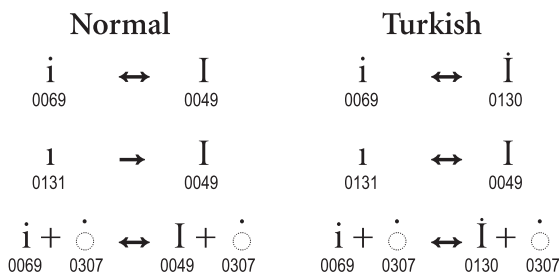
Figure 5-15 shows the lowercase mapping for Turkish *i*.

Figure 5-15. Lowercase Mapping for Turkish I
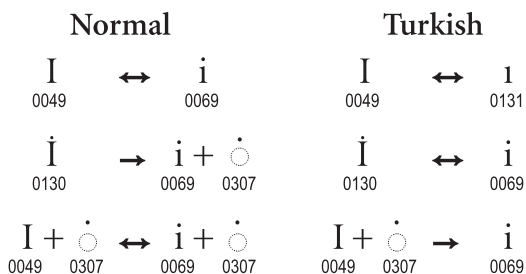
In both of the Turkish case mapping figures, a mapping with a double-sided arrow round-trips—that is, the opposite case mapping results in the original sequence. A mapping with a single-sided arrow does not round-trip.

Caseless Characters. Because many characters are really caseless (most of the IPA block, for example) and have no matching uppercase, the process of uppercasing a string does *not* mean that it will no longer contain any lowercase letters.

German sharp s. The German *sharp s* character has several complications in case mapping. Not only does its uppercase mapping expand in length, but its default case-pairings are asymmetrical. The default case mapping operations follow standard German orthography, which uses the string "SS" as the regular uppercase mapping for U+00DF ß LATIN SMALL LETTER SHARP S. In contrast, the alternate, single character uppercase form, U+1E9E LATIN CAPITAL LETTER SHARP S, is intended for typographical representations of signage and uppercase titles, and in other environments where users require the *sharp s* to be preserved in uppercase. Overall, such usage is uncommon. Thus, when using the default Unicode casing operations, *capital sharp s* will lowercase to *small sharp s*, but not vice versa: *small sharp s* uppercases to "SS", as shown in *Figure 5-16*. A tailored casing operation is needed in circumstances requiring *small sharp s* to uppercase to *capital sharp s*.

Figure 5-16. Casing of German Sharp S
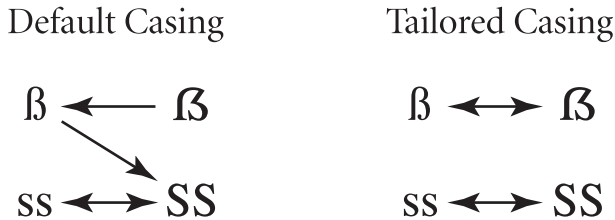
Additional language-specific or orthography-specific contexts and casing behavior is specified in the Unicode Common Locale Data Repository (CLDR), https://cldr.unicode.org.

5.18.3 Reversibility

No casing operations are reversible. For example:

toUppercase(toLowercase("John Brown")) → "JOHN BROWN"

toLowercase(toUppercase("John Brown")) → "john brown"

There are even single words like *vederLa* in Italian or the name *McGowan* in English, which are neither upper-, lower-, nor titlecase. This format is sometimes called *inner-caps*—or more informally *camelcase*—and it is often used in programming and in Web names. Once the string "McGowan" has been uppercased, lowercased, or title-cased, the original cannot be recovered by applying another uppercase, lowercase, or titlecase operation. There are also single characters that do not have reversible mappings, such as the Greek sigmas.

For word processors that use a single command-key sequence to toggle the selection through different casings, it is recommended to save the original string and return to it via the sequence of keys. The user interface would produce the following results in response to a series of command keys. In the following example, notice that the original string is restored every fourth time.

1. The quick brown

2. THE QUICK BROWN

3. the quick brown

4. The Quick Brown

5. The quick brown (repeating from here on)

Uppercase, titlecase, and lowercase can be represented in a word processor by using a character style. Removing the character style restores the text to its original state. However, if this approach is taken, any spell-checking software needs to be aware of the case style so that it can check the spelling against the actual appearance.

5.18.4 Caseless Matching

Caseless matching is implemented using *case folding*, which is the process of mapping characters of different case to a single form, so that case differences in strings are erased. Case folding allows for fast caseless matches in lookups because only binary comparison is required. It is more than just conversion to lowercase. For example, it correctly handles cases such as the Greek sigma, so that "όσος" and "ΟΣΟΣ" will match.

Normally, the original source string is not replaced by the folded string because that substitution may erase important information. For example, the name "Marco di Silva" would be folded to "marco di silva," losing the information regarding which letters are capitalized. Typically, the original string is stored along with a case-folded version for fast comparisons.

The CaseFolding.txt file in the Unicode Character Database is used to perform case folding (which is locale-independent). This file is generated from the case mappings in the Unicode Character Database, using both the single-character mappings and the multicharacter mappings. It folds all characters having different case forms together into a common form. To compare two strings for caseless matching, one can fold each string using this data and then use a binary comparison.

Case folding logically involves a set of equivalence classes constructed from the Unicode Character Database case mappings as follows.

For each character X in Unicode, apply the following rules in order:

R1 **If X is already in an equivalence class, continue to the next character. Otherwise, form a new equivalence class and add X.**

R2 **Add any other character that uppercases, lowercases, or titlecases to anything in the equivalence class.**

R3 **Add any other characters to which anything in the equivalence class uppercases, lowercases, or titlecases.**

R4 **Repeat R2 and R3 until nothing further is added.**

R5 **From each class, one representative element (a single lowercase letter where possible) is chosen to be the common form.**

For rule R5, it is preferable to choose a single lowercase letter for the common form, but this is not possible in all instances. For case folding of Cherokee letters, for example, a single *uppercase* letter must be chosen instead, because the uppercase letters for Cherokee were encoded in an earlier version of the Unicode Standard, and the lowercase letters were encoded in a later version. This choice is required to keep case folding stable across Unicode versions.

Each equivalence class is completely disjoint from all the others, and every Unicode character is in one equivalence class. CaseFolding.txt thus contains the mappings from other characters in the equivalence classes to their common forms. As an exception, the case foldings for dotless i and dotted I do not follow the derivation algorithm for all other case foldings. Instead, their case foldings are hard-coded in the derivation for best default matching behavior. There are alternate case foldings for these characters, which can be used for case folding for Turkic languages. However, the use of those alternate case foldings does not maintain canonical equivalence. Furthermore, it is often undesirable to have differing behavior for caseless matching. Because language information is often not available when caseless matching is applied to strings, it also may not be clear which alternate to choose.

The Unicode case folding algorithm is defined to be simpler and more efficient than case mappings. It is context-insensitive and language-independent (except for the optional, alternate Turkic case foldings). As a result, there are a few rare cases where a caseless match does not match pairs of strings as expected; the most notable instance of this is for Lithuanian. In Lithuanian typography for dictionary use, an "i" retains its dot when a grave, acute, or tilde accent is placed above it. This convention is represented in Unicode by using an explicit combining dot above, occurring in sequence between the "i" and the respective accent. (See *Figure 7-2*.) When case folded using the default case folding algorithm, strings containing these sequences will still contain the combining dot above. In the unusual situation where case folding needs to be tailored to provide for these special Lithuanian dictionary requirements, strings can be preprocessed to remove any combining dot above char-

acters occurring between an "i" and a subsequent accent, so that the folded strings will match correctly.

Where case distinctions are not important, other distinctions between Unicode characters (in particular, compatibility distinctions) are generally ignored as well. In such circumstances, text can be normalized to Normalization Form NFKC or NFKD after case folding, thereby producing a normalized form that erases both compatibility distinctions and case distinctions. However, such normalization should generally be done only on a restricted repertoire, such as identifiers (alphanumerics). See Unicode Standard Annex #15, "Unicode Normalization Forms," and Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax," for more information. For a summary, see "Equivalent Sequences" in *Section 2.2, Unicode Design Principles*.

Caseless matching is only an approximation of the language-specific rules governing the strength of comparisons. Language-specific case matching can be derived from the collation data for the language, where only the first- and second-level differences are used. For more information, see Unicode Technical Standard #10, "Unicode Collation Algorithm."

In most environments, such as in file systems, text is not and cannot be tagged with language information. In such cases, the language-specific mappings *must not* be used. Otherwise, data structures such as B-trees might be built based on one set of case foldings and used based on a different set of case foldings. This discrepancy would cause those data structures to become corrupt. For such environments, a constant, language-independent, default case folding is required.

Stability. The definition of case folding is guaranteed to be stable, in that any string of characters case folded according to these rules will remain case folded in Version 5.0 or later of the Unicode Standard. To achieve this stability, there are constraints on additions of case pairs for existing encoded characters. Typically, no new lowercase character will be added to the Unicode Standard as a casing pair of an existing upper- or titlecase character that does not already have a lowercase pair. In exceptional circumstances, where lowercase characters must be added to the standard in a later version than the version in which the corresponding uppercase characters were encoded, such lowercase characters can only be defined as new case pairs with a corresponding change to case folding to ensure that they *case fold* to the old uppercase letters. See the subsection "Policies" in *Appendix B.3, Other Unicode Online Resources*.

5.18.5 Normalization and Casing

Casing operations as defined in *Section 3.13, Default Case Algorithms* are not guaranteed to preserve Normalization Forms. That is, some strings in a particular Normalization Form (for example, NFC) will no longer be in that form after the casing operation is performed. Consider the strings shown in the example in *Table 5-5*.

Table 5-5. Casing and Normalization in Strings

Original (NFC)	ǰ ̣	U+01F0 LATIN SMALL LETTER J WITH CARON U+0323 COMBINING DOT BELOW
Uppercased	J ̌ ̣	U+004A LATIN CAPITAL LETTER J U+030C COMBINING CARON U+0323 COMBINING DOT BELOW
Uppercased NFC	J ̣ ̌	U+004A LATIN CAPITAL LETTER J U+0323 COMBINING DOT BELOW U+030C COMBINING CARON

The original string is in Normalization Form NFC format. When uppercased, the *small j with caron* turns into an *uppercase J* with a separate *caron*. If followed by a combining mark below, that sequence is not in a normalized form. The combining marks have to be put in canonical order for the sequence to be normalized.

If text in a particular system is to be consistently normalized to a particular form such as NFC, then the casing operators should be modified to normalize after performing their core function. The actual process can be optimized; there are only a few instances where a casing operation causes a string to become denormalized. If a system specifically checks for those instances, then normalization can be avoided where not needed.

Normalization also interacts with case folding. For any string X, let Q(X) = NFC(toCasefold(NFD(X))). In other words, Q(X) is the result of normalizing X, then case folding the result, then putting the result into Normalization Form NFC format. Because of the way normalization and case folding are defined, Q(Q(X)) = Q(X). Repeatedly applying Q does not change the result; case folding is *closed* under canonical normalization for either Normalization Form NFC or NFD.

Case folding is not, however, closed under compatibility normalization for either Normalization Form NFKD or NFKC. That is, given R(X) = NFKC(toCasefold(NFD(X))), there are some strings such that R(R(X)) ≠ R(X). NFKC_Casefold, a derived property, is closed under both case folding and NFKC normalization. The property values for NFKC_Casefold are found in DerivedNormalizationProps.txt in the Unicode Character Database.

5.19 Mapping Compatibility Variants

Identifying one character as a compatibility variant of another character (or sequence of characters) suggests that in many circumstances the first can be remapped to the second without the loss of any textual information other than formatting and layout. (See *Section 2.3, Compatibility Characters*.)

Such remappings or foldings can be done in different ways. In the case of compatibility decomposable characters, remapping occurs as a result of normalizing to the NFKD or NFKC forms defined by Unicode Normalization. Other compatibility characters which are not compatibility decomposable characters may be remapped by various kinds of folding; for example, Kangxi radical symbols in the range U+2F00..U+2FDF might be substituted by the corresponding CJK unified ideographs of the same appearance.

However, such remapping should not be performed indiscriminately, because many of the compatibility characters are included in the standard precisely to allow systems to maintain one-to-one mappings to other existing character encoding standards. In such cases, a remapping would lose information that is important to maintaining some distinction in the original encoding.

Thus an implementation must proceed with due caution—replacing a character with its compatibility decomposition or otherwise folding compatibility characters together with ordinary Unicode characters may change not only formatting information, but also other textual distinctions on which some other process may depend.

In many cases there exists a visual relationship between a compatibility character and an ordinary character that is akin to a font style or directionality difference. Replacing such characters with unstyled characters could affect the meaning of the text. Replacing them with rich text would preserve the meaning for a human reader, but could cause some programs that depend on the distinction to behave unpredictably. This issue particularly affects compatibility characters used in mathematical notation. For more discussion of these issues, see the W3C specification, "Unicode in XML and other Markup Languages," and Unicode Technical Report #25, "Unicode Support for Mathematics."

In other circumstances, remapping compatibility characters can be very useful. For example, transient remapping of compatibility decomposable characters using NFKC or NFKD normalization forms is very useful for performing "loose matches" on character strings. See also Unicode Technical Standard #10, "Unicode Collation Algorithm," for the role of compatibility character remapping when establishing collation weights for Unicode strings.

Confusables. The visual similarities between compatibility variants and ordinary characters can make them confusable with other characters, something that can be exploited in possible security attacks. Compatibility variants should thus be avoided in certain usage domains, such as personal or network identifiers. The usual practice for avoiding compatibility variants is to restrict such strings to those already in Normalization Form NFKC; this practice eliminates any compatibility decomposable characters. Compatibility decomposable characters can also be remapped on input by processes handling personal or network identifiers, using Normalization Form NFKC.

This general implementation approach to the problems associated with visual similarities among compatibility variants, by focusing first on the remapping of compatibility decomposable characters, is useful for two reasons. First, the large majority of compatibility variants are in fact also compatibility decomposable characters, so this approach deals with the biggest portion of the problem. Second, it is simply and reproducibly implementable in terms of a well-defined Unicode Normalization Form.

Extending restrictions on usage to other compatibility variants is more problematical, because there is no exact specification of which characters are compatibility variants. Furthermore, there may be valid reasons to restrict usage of certain characters which may be visually confusable or otherwise problematical for some process, even though they are not generally considered to be compatibility variants. Best practice in such cases is to depend on carefully constructed and justified lists of confusable characters.

For more information on security implications and a discussion of confusables, see Unicode Technical Report #36, "Unicode Security Considerations" and Unicode Technical Standard #39, "Unicode Security Mechanisms."

5.20 Unicode Security

It is sometimes claimed that the Unicode Standard poses new security issues. Some of these claims revolve around unique features of the Unicode Standard, such as its encoding forms. Others have to do with generic issues, such as character spoofing, which also apply to any other character encoding, but which are seen as more severe threats when considered from the point of view of the Unicode Standard.

This section examines some of these issues and makes some implementation recommendations that should help in designing secure applications using the Unicode Standard.

Alternate Encodings. A basic security issue arises whenever there are alternate encodings for the "same" character. In such circumstances, it is always possible for security-conscious modules to make different assumptions about the representation of text. This conceivably can result in situations where a security watchdog module of some sort is screening for prohibited text or characters, but misses the same characters represented in an alternative form. If a subsequent processing module then treats the alternative form as if it were what the security watchdog was attempting to prohibit, one potentially has a situation where a hostile outside process can circumvent the security software. Whether such circumvention can be exploited in any way depends entirely on the system in question.

Some earlier versions of the Unicode Standard included enough leniency in the definition of the UTF-8 encoding form, particularly regarding the so-called *non-shortest form*, to raise questions about the security of applications using UTF-8 strings. However, the conformance requirements on UTF-8 and other encoding forms in the Unicode Standard have been tightened so that no encoding form now allows any sort of alternate representation, including non-shortest form UTF-8. Each Unicode code point has a single, unique encoding in any particular Unicode encoding form. Properly coded applications should not be subject to attacks on the basis of code points having multiple encodings in UTF-8 (or UTF-16).

However, another level of alternate representation has raised other security questions: the canonical equivalences between precomposed characters and combining character sequences that represent the same abstract characters. This is a different kind of alternate representation problem—not one of the encoding forms per se, but one of visually identical characters having two distinct representations (one as a single encoded character and one as a sequence of base form plus combining mark, for example). The issue here is different from that for alternate encodings in UTF-8. Canonically equivalent representations for the "same" string are perfectly valid and expected in Unicode. The conformance requirement, however, is that conforming implementations cannot be *required* to make an interpretation distinction between canonically equivalent representations. The way for a security-conscious application

to guarantee this is to carefully observe the normalization specifications (see Unicode Standard Annex #15, "Unicode Normalization Forms") so that data is handled consistently in a normalized form.

Spoofing. Another security issue is *spoofing*, meaning the deliberate misspelling of a domain name, or user name, or other string in a form designed to trick unwary users into interacting with a hostile website as if it was a trusted site (or user). In this case, the confusion is not at the level of the software process handling the code points, but rather in the human end users, who see one character but mistake it for another, and who then can be fooled into doing something that will breach security or otherwise result in unintended results.

To be effective, spoofing does not require an exact visual match—for example, using the digit "1" instead of the letter "l". The Unicode Standard contains many *confusables*—that is, characters whose glyphs, due to historical derivation or sheer coincidence, resemble each other more or less closely. Certain security-sensitive applications or systems may be vulnerable due to possible misinterpretation of these confusables by their users.

Many legacy character sets, including ISO/IEC 8859-1 or even ASCII, also contain confusables, albeit usually far fewer of them than in the Unicode Standard simply because of the sheer scale of Unicode. The legacy character sets all carry the same type of risks when it comes to spoofing, so there is nothing unique or inadequate about Unicode in this regard. Similar steps will be needed in system design to assure integrity and to lessen the potential for security risks, no matter which character encoding is used.

The Unicode Standard encodes characters, not glyphs, and it is impractical for many reasons to try to avoid spoofing by simply assigning a single character code for every possible confusable glyph among all the world's writing systems. By unifying an encoding based strictly on appearance, many common text-processing tasks would become convoluted or impossible. For example, Latin B and Greek Beta B look the same in most fonts, but lower-case to two different letters, Latin b and Greek beta β, which have very distinct appearances. A simplistic fix to the confusability of Latin B and Greek Beta would result in great difficulties in processing Latin and Greek data, and in many cases in data corruptions as well.

Because all character encodings inherently have instances of characters that might be confused with one another under some conditions, and because the use of different fonts to display characters might even introduce confusions between characters that the designers of character encodings could not prevent, character spoofing must be addressed by other means. Systems or applications that are security-conscious can test explicitly for known spoofings, such as "MICROS0FT," "A0L," or the like (substituting the digit "0" for the letter "O"). Unicode-based systems can provide visual clues so that users can ensure that labels, such as domain names, are within a single

script to prevent cross-script spoofing. However, provision of such clues is clearly the responsibility of the system or application, rather than being a security condition that could be met by somehow choosing a "secure" character encoding that was not subject to spoofing. No such character encoding exists.

Unicode Standard Annex #24, "Unicode Script Property," presents a classification of Unicode characters by script. By using such a classification, a program can check that labels consist only of characters from a given script or characters that are expected to be used with more than one script (such as the "Common" or "Inherited" script names defined in Unicode Standard Annex #24, "Unicode Script Property"). Because cross-script names may be legitimate, the best method of alerting a user might be to highlight any unexpected boundaries between scripts and let the user determine the legitimacy of such a string explicitly.

For further discussion of security issues, see Unicode Technical Report #36, "Unicode Security Considerations," and Unicode Technical Standard #39, "Unicode Security Mechanisms."

5.21 Ignoring Characters in Processing

The majority of encoded characters in the Unicode Standard are ordinary graphic characters. However, the standard also includes a significant number of special-use characters. For example, format characters (General_Category = Cf) are often defined to have very particular effects in text processing. These effects may impact one kind of text process, but be completely irrelevant for other text processes. Format characters also typically have no visible display of their own, but may impact the display of neighboring graphic characters. Technically, variation selectors are not format characters, but combining marks. However, variation selectors and other "invisible" combining marks also have special behavior in text processing.

Other sections of the Unicode Standard specify the intended effects of such characters in detail. See, for example, *Section 23.2, Layout Controls* and *Section 23.4, Variation Selectors*. This section, on the other hand, approaches the issue by discussing which kinds of format characters (and other characters) are *ignored* for different kinds of text processes, and providing pointers to related implementation guidelines.

How these kinds of special-use characters are displayed or *not* displayed in various contexts is of particular importance. Many have no inherent display of their own, so pose questions both for normal rendering for display and for fallback rendering. Because of this, a particularly detailed discussion of ignoring characters for display can be found toward the end of this section.

5.21.1 Characters Ignored in Text Segmentation

Processing for text segmentation boundaries generally ignores certain characters which are irrelevant to the determination of those boundaries. The exact classes of characters depend on which type of text segmentation is involved.

When parsing grapheme cluster boundaries, characters used to extend grapheme clusters are ignored for boundary determination. These include nonspacing combining marks and enclosing marks, as well as U+200C ZERO WIDTH NON-JOINER. The exact list of characters involved is specified by the property value: Grapheme_Cluster_Break = Extend. U+200D ZERO WIDTH JOINER requires special handling, particularly for emoji sequences.

When parsing word or sentence boundaries, the set of characters which are ignored for boundary determination is enlarged somewhat, to include spacing combining marks and most format characters. For word breaking, the exact list of characters is specified by two property values: Word_Break = Extend or Word_Break = Format. For sentence breaking, the corresponding property values are: Sentence_Break = Extend or Sentence_Break = Format.

For a detailed discussion of text segmentation, see Unicode Standard Annex #29, "Unicode Text Segmentation." In particular, see Section 6.2, *Replacing Ignore Rules,* in that annex, for implementation notes about the rules which ignore classes of characters for segmentation.

5.21.2 Characters Ignored in Line Breaking

Most control characters and format characters are ignored for line break determination, and do not contribute to line width. The Unicode Line Breaking Algorithm handles this class of characters by giving them the same Line_Break property value as combining marks: Line_Break = CM. For a detailed discussion, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm."

When expanding or compressing intercharacter space, as part of text justification and determination of line breaks, the presence of U+200B ZERO WIDTH SPACE or U+2060 WORD JOINER is generally ignored. There are, however, occasional exceptions. See, for example, the discussion of "Thai-style" letter spacing in *Section 23.2, Layout Controls*.

5.21.3 Characters Ignored in Cursive Joining

U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER are format controls specifically intended to influence cursive joining. However, there are other format controls which are explicitly ignored when processing text for cursive joining. In particular, U+2060 WORD JOINER, U+FEFF ZERO WIDTH NO-BREAK SPACE, and U+200B ZERO WIDTH SPACE influence text segmentation and line breaking, but should be ignored for cursive joining. U+034F COMBINING GRAPHEME JOINER is also ignored for cursive joining.

More generally, there is a broad class of characters whose occurrence in a string should be ignored when calculating cursive connections between adjacent letters subject to cursive joining. This class is defined by the property value, Joining_Type = Transparent, and includes all nonspacing marks and most format characters other than ZWNJ and ZWJ. See the detailed discussion of cursive joining in *Section 23.2, Layout Controls*.

5.21.4 Characters Ignored in Identifiers

Characters with the property Default_Ignorable_Code_Point (DI) are generally not recommended for inclusion in identifiers. Such characters include many (but not all) format characters, as well as variation selectors. Exceptions are the cursive joining format characters, U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER, which in limited circumstances may be used to make visual distinctions deemed necessary for identifiers.

There are several possible approaches for ensuring that characters with DI = True are not significant for comparison of identifiers. A strict formal syntax definition may simply prohibit their inclusion in identifier strings altogether. However, comparison of identifiers often involves a folding operation, such as case folding. In applications which implement identifier folding based on the toNFKC_CaseFold transformation, DI = True characters are removed from a string by that transformation. With such an approach, DI= True characters can be said to be "ignored" in identifier comparison, and their presence or absence in a given identifier string is irrelevant to the comparison. See Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax," for a detailed discussion of normalization and case folding of identifiers and of the handling of format characters in identifiers.

5.21.5 Characters Ignored in Searching and Sorting

Searching and string matching is another context in which particular characters may be ignored. Typically, users expect that certain characters, such as punctuation, will be ignored when looking for string matches against a target string, or they expect that certain character distinctions, such as case differences, will be ignored. Exact binary string comparisons in such circumstances produce the wrong results.

At its core, sorting string data involves using a string matching algorithm to determine which strings count as equal. In any comparison of strings which do not count as equal, sorting additionally requires the ability to determine which string comes before and which after in the collation order. It is important to have a well-defined concept of which characters "do not make a difference," and are thus ignored for the results of the sorting.

Some Unicode characters almost never make a significant difference for searching, string matching, and sorting. For example, U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER may impact cursive joining or ligature formation, but are not intended to represent semantic differences between strings. At a first level of approximation, most Unicode format controls should be ignored for searching and sorting. However, there is no unique way to use Unicode character properties to devise an exact list of which characters should always be ignored for searching and sorting, in part because the criteria for any particular search or sort can vary so widely.

The Unicode algorithm which addresses this issue generically is defined in Unicode Technical Standard #10, "Unicode Collation Algorithm." The Default Unicode Collation Element Table (DUCET), documented in that standard, provides collation weights for all Unicode characters; many of those weights are set up so that the characters will be ignored by default for sorting. A string matching algorithm can also be based on the weights in that table. Additionally, the UCA provides options for ignoring *distinctions* between related characters, such as uppercase versus lowercase letters, or letters with or without accents. The UCA provides a mechanism to tailor

the DUCET. This mechanism not only enables the general algorithm to support different tailored tables which allow for language-specific orderings of characters, it also makes it possible to specify very precisely which characters should or should not be ignored for any particular search or sort.

5.21.6 Characters Ignored for Display

There are two distinct cases to consider when determining whether a particular character should be "ignored" for display. The first case involves normal rendering, when a process supports the character in question. The second case involves fallback rendering, when the character in question is outside the repertoire which can be supported for normal rendering, so that a fallback to exceptional rendering for unknown characters is required.

In this discussion, "display" is used as shorthand for the entire text rendering process, which typically involves a combination of rendering software and font definition. Having a display glyph for a character defined in a font is not sufficient to render it for screen display or for printing; rendering software is involved as well. On the other hand, fonts may contain complex rendering logic which contributes to the text rendering process. This discussion is not meant to preclude any particular approach to the design of a full text rendering process. A phrase such as, "a font displays a glyph for the character," or "a font displays no glyph for the character," is simply a general way of describing the intended display outcome for rendering that character.

Normal Rendering. Many characters, including format characters and variation selectors, have no visible glyph or advance width directly associated with them. Such characters without glyphs are typically shown in the code charts with special display glyphs using a dotted box and a mnemonic label. (See *Section 24.1, Character Names List*, for code chart display conventions.) Outside of the particular context of code chart display, a font will typically display no glyph for such characters. However, it is not unusual for format characters and variation selectors to have a visible effect on other characters in their vicinity. For example, ZWJ and ZWNJ may affect cursive joining or the appearance of ligatures. A variation selector may change the choice of glyph for display of the base character it follows. In such cases, even though the format character or variation selector has no visible glyph of its own, it would be inappropriate to say that it is *ignored* for display, because the intent of its use is to change the display in some visible way. Additional cases where a format character has no glyph, but may otherwise affect display include:

- Bidirectional format characters do not affect the glyph forms of displayed characters, but may cause significant rearrangements of spans of text in a line.

- U+00AD ⟦SHY⟧ soft hyphen has a null default appearance in the middle of a line: the appearance of "ther⟦SHY⟧apist" is simply "therapist"—no visible glyph. In line

break processing, it indicates a possible intraword break. At any intraword break that is used for a line break—whether resulting from this character or by some automatic process—a hyphen glyph (perhaps with spelling changes) or some other indication can be shown, depending on language and context.

In other contexts, a format character may have no visible effect on display at all. For example, a ZWJ might occur in text between two characters which are not subject to cursive joining and for which no ligature is available or appropriate: <x, ZWJ, x>. In such a case, the ZWJ simply has no visible effect, and one can meaningfully say that it is *ignored* for display. Another example is a variation selector following a base character for which no standardized or registered variation sequence exists. In that case, the variation selector has no effect on the display of the text.

Finally, there are some format characters whose function is not intended to affect display. U+200B ZERO WIDTH SPACE affects word segmentation, but has no visible display. U+034F COMBINING GRAPHEME JOINER is likewise always ignored for display. Additional examples include:

- U+2060 ⒲ WORD JOINER does not produce a visible change in the appearance of surrounding characters; instead, its only effect is to indicate that there should be no line break at that point.

- U+2061 ⒡⒪ FUNCTION APPLICATION has no effect on the text display and is used only in internal mathematical expression processing.

Disruption of Tightly Defined Sequences. In some instances, the mere presence of an otherwise invisible character may affect the display of tightly defined sequences. A fairly obvious example would be the insertion of a U+200B ZERO WIDTH SPACE or a U+2060 WORD JOINER into the middle of a combining character sequence. Such an insertion formally breaks the combining character sequence, which has a tightly defined normative syntax. (See D56 in *Section 3.6, Combination*.) The insertion may then result in unexpected display results, including the appearance of dotted circles or other visual disruption.

The correct use of U+2044 FRACTION SLASH or various prepended concatenation marks (see *Figure 9-7*) also depends on a tightly constrained syntax for neighboring characters. For example, an implementation that supports the fraction slash character can take a preceding string of digits and a succeeding string of digits, and reformat them as the numerator and denominator of a vulgar fraction for display. However, the insertion of any invisible format character into those strings of digits would break the sequences of the digits and thus result in an unexpected display. A similar outcome can be anticipated for the insertion of invisible format characters into any sequence of digits following a prepended concatenation mark.

The principle is that while many format characters have no visible glyphs and are usually ignored for display, that does not obligate implementations to accommodate

their occurrence in *any* position in text without disruption of display, particularly when they interrupt the syntax of otherwise tightly defined sequences with specific interpretations in the standard.

Show Hidden Mode. The fact that variation selectors and most format characters have no visible glyphs does not mean that such characters must always be invisible. An implementation can, for example, show a visible glyph on request, such as in a "Show Hidden" mode. A particular use of a "Show Hidden" mode is to display a visible indication of misplaced or ineffectual format characters. For example, a sequence of two adjacent joiners, <…, ZWJ, ZWJ, …>, is a case where the extra ZWJ should have no effect.

Whitespace Characters. Format characters with no visible glyphs are different from space characters. Space characters, such as U+0020 SPACE, are classified as graphic characters. Although they do not have *visible* glyphs for display, they have advance widths. Technically, that counts as a "glyph" in a font—it is simply a blank glyph "with no pixels turned on." Like other graphic characters, a space character can be visibly selected in text. Line separation characters, such as the *carriage return*, do not clearly exhibit their advance width, because they always occur at the end of a line, but most implementations give them a visible advance width when they are selected. Hence, they are classed together with space characters; both are given the White_Space property. Whitespace characters are not considered to be ignored for display.

Fallback Rendering. Fallback rendering occurs when a text process needs to display a character or sequence of characters, but lacks the rendering resources to display that character correctly. The typical situation results from having text to display without an appropriate font covering the repertoire of characters used in that text. The recommended behavior for display in such cases is to fall back to some visible, but generic, glyph display for graphic characters, so that at least it is clear that there are characters present—and usually, how many are present. (See *Section 5.3, Unknown and Missing Characters*.) However, variation selectors and some format characters are special—it is not appropriate for fallback rendering to display them with visible glyphs. This is illustrated by the following examples.

First consider an ordinary graphic character. For example, if an implementation does not support U+0915 क DEVANAGARI LETTER KA, it should not ignore that character for display. Displaying *nothing* would give the user the impression that the character does not occur in the text at all. The recommendation in that case is to display a "last-resort" glyph or a visible "missing glyph" box, instead.

Contrast that with the typical situation for a format character, such as ZWJ. If an implementation does not support that character at all, the best practice is to ignore it completely for display, without showing a last-resort glyph or a visible box in its place. This is because even for normal rendering a ZWJ is invisible—its visible effects

are on other characters. When an implementation does not support the behavior of a ZWJ, it has no way of showing the effects on neighboring characters.

Default Ignorable Code Point. The list of characters which should be ignored for display in fallback rendering is given by a character property: Default_Ignorable_Code_Point (DI). Those characters include almost all format characters, all variation selectors, and a few other exceptional characters, such as Hangul fillers. The exact list is defined in DerivedCoreProperties.txt in the Unicode Character Database.

The Default_Ignorable_Code_Point property is also given to certain ranges of code points: U+2060..U+206F, U+FFF0..U+FFF8, and U+E0000..U+E0FFF, including any *unassigned* code points in those ranges. These ranges are designed and reserved for future encoding of format characters and similar special-use characters, to allow a certain degree of forward compatibility. Implementations which encounter unassigned code points in these ranges should ignore them for display in fallback rendering.

Surrogate code points, private-use characters, and control characters are not given the Default_Ignorable_Code_Point property. To avoid security problems, such characters or code points, when not interpreted and not displayable by normal rendering, should be displayed in fallback rendering with a fallback glyph, so that there is a visible indication of their presence in the text. For more information, see Unicode Technical Report #36, "Unicode Security Considerations."

A small number of format characters (General_Category = Cf) are also not given the Default_Ignorable_Code_Point property. This may surprise implementers, who often assume that all format characters are generally ignored in fallback display. The exact list of these exceptional format characters can be found in the Unicode Character Database. There are, however, three important sets of such format characters to note:

- prepended concatenation marks
- interlinear annotation characters
- Egyptian hieroglyph format controls

The prepended concatenation marks always have a visible display. See "Prepended Concatenation Marks" in *Section 23.2, Layout Controls* for more discussion of the use and display of these signs.

The other two notable sets of format characters that exceptionally are not ignored in fallback display consist of the interlinear annotation characters, U+FFF9 INTERLINEAR ANNOTATION ANCHOR through U+FFFB INTERLINEAR ANNOTATION TERMINATOR, and the Egyptian hieroglyph format controls, U+13430 EGYPTIAN HIEROGLYPH VERTICAL JOINER through U+1343F EGYPTIAN HIEROGLYPH END WALLED ENCLOSURE. These characters should have a visible glyph display for fall-

back rendering, because if they are not displayed, it is too easy to misread the resulting displayed text. See "Annotation Characters" in *Section 23.8, Specials*, as well as *Section 11.4, Egyptian Hieroglyphs* for more discussion of the use and display of these characters.

5.22 U+FFFD Substitution in Conversion

When converting text from one character encoding to another, a conversion algorithm may encounter unconvertible code units. This is most commonly caused by some sort of corruption of the source data, so that it does not correctly follow the specification for that character encoding. Examples include dropping a byte in a multibyte encoding such as Shift-JIS, improper concatenation of strings, a mismatch between an encoding declaration and actual encoding of text, use of non-shortest form for UTF-8, and so on.

When a conversion algorithm encounters such unconvertible data, the usual practice is either to throw an exception or to use a defined substitution character to represent the unconvertible data. In the case of conversion to one of the encoding forms of the Unicode Standard, the substitution character is defined as U+FFFD REPLACEMENT CHARACTER.

For conversion *between* different encoding forms of the Unicode Standard, "U+FFFD Substitution of Maximal Subparts" in *Section 3.9, Unicode Encoding Forms* defines a practice for the use of U+FFFD which is consistent with the W3C standard for encoding. It is useful to apply the same practice to the conversion from non-Unicode encodings to an encoding form of the Unicode Standard.

This practice is more secure because it does not result in the conversion consuming parts of valid sequences as though they were invalid. It also guarantees at least one replacement character will occur for each instance of an invalid sequence in the original text. Furthermore, this practice can be defined consistently for better interoperability between different implementations of conversion.

For full consistency, it is important for conversion implementations to agree on 1) the exact set of well-formed sequences for the source encoding, 2) all of the mappings for valid sequences, and 3) the details of the practice for handling ill-formed sequences.

Chapter 6

Writing Systems and Punctuation

This chapter begins the portion of the Unicode Standard devoted to the detailed description of each script or other related group of Unicode characters. Each of the subsequent chapters presents a historically or geographically related group of scripts. This chapter presents a general introduction to writing systems, explains how they can be used to classify scripts, and then presents a detailed discussion of punctuation characters that are shared across scripts.

Scripts and Blocks. The codespace of the Unicode Standard is divided into subparts called *blocks* (see D10b in *Section 3.4, Characters and Encoding*). Character blocks generally contain characters from a single script, and in many cases, a script is fully represented in its block; however, some scripts are encoded using several blocks, which are not always adjacent. Discussion of scripts and other groups of characters are structured by blocks. Corresponding subsection headers identify each block and its associated range of Unicode code points. The Unicode code charts are also organized by blocks.

Scripts and Writing Systems. There are many different kinds of writing systems in the world. Their variety poses some significant issues for character encoding in the Unicode Standard as well as for implementers of the standard. Those who first approach the Unicode Standard without a background in writing systems may find the huge list of scripts bewilderingly complex. Therefore, before considering the script descriptions in detail, this chapter first presents a brief introduction to the types of writing systems. That introduction explains basic terminology about scripts and character types that will be used again and again when discussing particular scripts.

Punctuation. The rest of this chapter deals with a special case: punctuation marks, which tend to be scattered about in different blocks and which may be used in common by many scripts. Punctuation characters occur in several widely separated places in the blocks, including Basic Latin, Latin-1 Supplement, General Punctuation, Supplemental Punctuation, and CJK Symbols and Punctuation. There are also occasional punctuation characters in blocks for specific scripts.

Most punctuation characters are intended for common usage with any script, although some of them are script-specific. Some scripts use both common and script-specific punctuation characters, usually as the result of recent adoption of standard Western punctuation marks. While punctuation characters vary in details of appearance and function between different languages and scripts, their overall purpose is shared: they serve to separate or otherwise organize units of text, such as sentences and phrases, thereby helping to clarify the meaning of the text. Certain punctuation characters also occur in mathematical and scientific formulae.

6.1 Writing Systems

This section presents a brief introduction to writing systems. It describes the different kinds of writing systems and relates them to the encoded scripts found in the Unicode Standard. This framework may help to make the variety of scripts, modern and historic, a little less daunting. The terminology used here follows that developed by Peter T. Daniels, a leading expert on writing systems of the world.

The term *writing system* has two mutually exclusive meanings in this standard. As used in this section, "writing system" refers to a way that families of scripts may be classified by how they represent the sounds or words of human language. For example, the writing system of the Latin script is alphabetic. In other places in the standard, "writing system" refers to the way a particular *language* is written. For example, the modern Japanese writing system uses four scripts: Han ideographs, Hiragana, Katakana and Latin (Romaji).

Alphabets. A writing system that consists of letters for the writing of both consonants and vowels is called an *alphabet*. The term "alphabet" is derived from the first two letters of the Greek script: *alpha, beta*. Consonants and vowels have equal status as letters in such a system. The Latin alphabet is the most widespread and well-known example of an alphabet, having been adapted for use in writing thousands of languages.

The correspondence between letters and sounds may be either more or less exact. Many alphabets do not exhibit a one-to-one correspondence between distinct sounds and letters or groups of letters used to represent them; often this is an indication of original spellings that were not changed as the language changed. Not only are many sounds represented by letter combinations, such as "th" in English, but the language may have evolved since the writing conventions were settled. Examples range from cases such as Italian or Finnish, where the match between letter and sound is rather close, to English, which has notoriously complex and arbitrary spelling.

Phonetic alphabets, in contrast, are used specifically for the precise transcription of the sounds of languages. The best known of these alphabets is the *International Phonetic Alphabet*, an adaptation and extension of the Latin alphabet by the addition of new letters and marks for specific sounds and modifications of sounds. Unlike normal alphabets, the intent of phonetic alphabets is that their letters exactly represent sounds. Phonetic alphabets are not used as general-purpose writing systems per se, but it is not uncommon for a formerly unwritten language to have an alphabet developed for it based on a phonetic alphabet.

Abjads. A writing system in which only consonants are indicated is an *abjad*. The main letters are all consonants (or long vowels), with other vowels either left out entirely or optionally indicated with the use of secondary marks on the consonants.

The Phoenician script is a prototypical abjad; a better-known example is the Arabic writing system. The term "abjad" is derived from the first four letters of the traditional order of the Arabic script: *alef, beh, jeem, dal*. Abjads are often, although not exclusively, associated with Semitic languages, which have word structures particularly well suited to the use of consonantal writing. Some abjads allow consonant letters to mark long vowels, as the use of *waw* and *yeh* in Arabic for /u:/ or /i:/.

Hebrew and Arabic are typically written without any vowel marking at all. The vowels, when they do occur in writing, are referred to as *points* or *harakat*, and are indicated by the use of diacritic dots and other marks placed above and below the consonantal letters.

Syllabaries. In a *syllabary*, each symbol of the system typically represents both a consonant and a vowel, or in some instances more than one consonant and a vowel. One of the best-known examples of a syllabary is Hiragana, used for Japanese, in which the units of the system represent the syllables *ka*, *ki*, *ku*, *ke*, *ko*, *sa*, *si*, *su*, *se*, *so*, and so on. In general parlance, the elements of a syllabary are not called *letters*, but rather *syllables*.This can lead to some confusion, however, because letters of alphabets and units of other writing systems are also used, singly or in combinations, to write syllables of languages. So in a broad sense, the term "letter" can be used to refer to the syllables of a syllabary.

In syllabaries such as Cherokee, Hiragana, Katakana, and Yi, each symbol has a unique shape, with no particular shape relation to any of the consonant(s) or vowels of the syllables. In other cases, however, the syllabic symbols of a syllabary are not atomic; they can be built up out of parts that have a consistent relationship to the phonological parts of the syllable. The best example of this is the Hangul writing system for Korean. Each Hangul syllable is made up of a part for the initial consonant (or consonant cluster), a part for the vowel (or diphthong), and an optional part for the final consonant (or consonant cluster). The relationship between the sounds and the graphic parts to represent them is systematic enough for Korean that the graphic parts collectively are known as *jamos* and constitute a kind of alphabet on their own.

The jamos of the Hangul writing system have another characteristic: their shapes are not completely arbitrary, but were devised with intentionally iconic shapes relating them to articulatory features of the sounds they represent in Korean. The Hangul writing system has thus also been classified as a *featural syllabary*.

Abugidas. *Abugidas* represent a kind of blend of syllabic and alphabetic characteristics in a writing system. The Ethiopic script is an abugida. The term "abugida" is derived from the first four letters of the Ethiopic script in the Semitic order: *alf, bet, gaml, dant*. The order of vowels (-ä -u -i -a) is that of the traditional vowel order in the first four columns of the Ethiopic syllable chart. Historically, abugidas spread

across South Asia and were adapted by many languages, often of phonologically very different types.

This process has also resulted in many extensions, innovations, and/or simplifications of the original patterns. The best-known example of an abugida is the Devanagari script, used in modern times to write Hindi and many other Indian languages, and used classically to write Sanskrit. See *Section 12.1, Devanagari*, for a detailed description of how Devanagari works and is rendered.

In an abugida, each consonant letter carries an inherent vowel, usually /a/. There are also vowel letters, often distinguished between a set of independent vowel letters, which occur on their own, and dependent vowel letters, or *matras*, which are subordinate to consonant letters. When a dependent vowel letter follows a consonant letter, the vowel overrides the inherent vowel of the consonant. This is shown schematically in *Figure 6-1*.

Figure 6-1. Overriding Inherent Vowels

ka + i → ki	ka + u → ku
ka + e → ke	ka + o → ko

Abugidas also typically contain a special element usually referred to as a *halant*, *virama*, or *killer*, which, when applied to a consonant letter with its inherent vowel, has the effect of *removing* the inherent vowel, resulting in a bare consonant sound.

In Brahmi-derived scripts, text often needs to be interpreted as a sequence of orthographic syllables, each of which is a two-dimensional visual arrangement of components that form a unit. At the core of an orthographic syllable is a base character, which can be a consonant, an independent vowel, a numeric character, or a ligature formed from base characters and other characters. Attached to this core may be dependent forms (such as half-forms, subjoined forms, *repha* forms, medial forms) of consonants or independent vowels, as well as *nukta* marks, *virama* marks, dependent vowel marks, register shifter marks, tone marks, final consonant marks, and other marks. It is common for different components of orthographic syllables to form ligatures. Orthographic syllables often do not correspond to phonological syllables; it is common for the final consonants of phonological syllables to become the base characters, or sometimes dependent forms, of subsequent orthographic syllables.

Because of legacy practice, three distinct approaches have been taken in the Unicode Standard for the encoding of abugidas: the Devanagari model, the Tibetan model, and the Thai model. The Devanagari model, used for most abugidas, represents text in primarily phonetic order and encodes a virama character that can combine with adjacent consonants to create conjunct forms. The Tibetan model also uses the primarily phonetic order, but its subjoined consonants are encoded directly rather than as virama-consonant sequences. The Thai model represents text in primarily

visual display order, based on the typewriter legacy; neither Thai nor the other scripts using this model have conjunct forms.

The Ethiopic script is traditionally analyzed as an abugida, because the base character for each consonantal series is understood as having an inherent vowel. However, Ethiopic lacks some of the typical features of Brahmi-derived scripts, such as halants and matras. Historically, it was derived from early Semitic scripts and in its earliest form was an abjad. In its traditional presentation and its encoding in the Unicode Standard, it is now treated more like a syllabary.

Logosyllabaries. The final major category of writing system is known as the *logosyllabary*. In a logosyllabary, the units of the writing system are used primarily to write words and/or morphemes of words, with some subsidiary usage to represent syllabic sounds per se.

The best example of a logosyllabary is the Han script, used for writing Chinese and borrowed by a number of other East Asian languages for use as part of their writing systems. The term for a unit of the Han script is *hànzì* 漢字 in Chinese, *kanji* 漢字 in Japanese, and *hanja* 漢字 in Korean. In many instances this unit also constitutes a word, but more typically, two or more units together are used to write a word.

The basic unit of a logosyllabary has variously been referred to as an *ideograph* (also *ideogram*), a *logograph* (also *logogram*), or a *sinogram*. Other terms exist as well, and especially for poorly understood or undeciphered writing systems, the units of writing may simply be called *signs*. Notionally, a logograph (or logogram) is a unit of writing which represents a word or morpheme, whereas an ideograph (or ideogram) is a unit of writing which represents an idea or concept. However, the lines between these terms are often unclear, and usage varies widely. The Unicode Standard makes no principled distinction between these terms, but rather follows the customary usage associated with a given script or writing system. For the Han script, the term *CJK ideograph* (or *Han ideograph*) is used.

There are a number of other historical examples of logosyllabaries, such as Tangut. They vary in the degree to which they combine logographic writing principles, where the symbols stand for morphemes or entire words, and syllabic writing principles, where the symbols come to represent syllables per se, divorced from their meaning as morphemes or words. In some notable instances, as for Sumero-Akkadian cuneiform, a logosyllabary may evolve through time into a syllabary or alphabet by shedding its use of logographs. In other instances, as for the Han script, the use of logographic characters is very well entrenched and persistent. However, even for the Han script a small number of characters are used purely to represent syllabic sounds, so as to be able to represent such things as foreign personal names and place names.

Egyptian hieroglyphs constitute another mixed example. The majority of the hieroglyphs are logographs, but Egyptian hieroglyphs also contain a well-defined subset

that functions as an alphabet, in addition to other signs that represent sequences of consonants. And some hieroglyphs serve as semantic determinatives, rather than logographs in their own right—a function which bears some comparison to the way radicals work in CJK ideographs. To simplify the overall typology of Unicode scripts, Egyptian hieroglyphs and other hieroglyphic systems are lumped together with true logosyllabaries such as Han, but there are many differences in detail. For more about Egyptian hieroglyphs, in particular, see *Section 11.4, Egyptian Hieroglyphs*.

The classification of a writing system is often rendered somewhat ambiguous by complications in the exact ways in which it matches up written elements to the phonemes or syllables of a language. For example, although Hiragana is classified as a syllabary, it does not always have an exact match between syllables and written elements. Syllables with long vowels are not written with a single element, but rather with a sequence of elements. Thus the syllable with a long vowel *kū* is written with two separate Hiragana symbols, {ku}+{u}.

There may also be complications when a writing system deviates from the historical model from which it derives. For example, Mahajani and Multani are both based on the Brahmi model, but are structurally simpler than an abugida. These writing systems do not contain a virama. They also do not have matras and consonant conjunct formation characteristic to abugidas. Instead, Mahajani and Multani behave respectively as an alphabet and an abjad, and are encoded and classified accordingly in the Unicode Standard.

Because of these kinds of complications, one must always be careful not to assume too much about the structure of a writing system from its nominal classification.

Typology of Scripts in the Unicode Standard. *Table 6-1* lists all of the scripts currently encoded in the Unicode Standard, showing the writing system type for each. The list is an approximate guide, rather than a definitive classification, because of the mix of features seen in many scripts. The writing systems for some languages may be quite complex, mixing more than one type of script together in a composite system. Japanese is the best example; it mixes a logosyllabary (Han), two syllabaries (Hiragana and Katakana), and one alphabet (Latin, for *romaji*). In some instances, there is not a one-to-one correspondence between a script and its type, because the type may depend on how the script is used. For example, the Sunuwar script is used as an alphabet by the community in Nepal, but is used as an abugida in Sikkim.

Table 6-1. Typology of Scripts in the Unicode Standard

Alphabets	Adlam, Armenian, Avestan, Bassa Vah, Carian, Caucasian Albanian, Coptic, Cyrillic, Deseret, Elbasan, Garay, Georgian, Glagolitic, Gothic, Greek, Hanifi Rohingya, Kayah Li, Latin, Lisu, Lycian, Lydian, Mahajani, Mandaic, Medefaidrin, Meroitic Cursive, Meroitic Hieroglyphs, Mongolian, Mro, Nag Mundari, N'Ko, Nyiakeng Puachue Hmong, Ogham, Ol Chiki, Ol Onal, Old Hungarian, Old Italic, Old Permic, Old Persian, Old Turkic, Osage, Osmanya, Pahawh Hmong, Pau Cin Hau, Runic, Shavian, Sunuwar, Tangsa, Thaana, Tifinagh, Todhri, Toto, Ugaritic, Vithkuqi, Wancho, Warang Citi, Yezidi
Abjads	Arabic, Chorasmian, Elymaic, Hatran, Hebrew, Imperial Aramaic, Inscriptional Pahlavi, Inscriptional Parthian, Manichaean, Multani, Nabataean, Old North Arabian, Old Sogdian, Old South Arabian, Old Uyghur, Palmyrene, Phoenician, Psalter Pahlavi, Samaritan, Sogdian, Syriac
Abugidas	Ahom, Balinese, Batak, Bengali, Bhaiksuki, Brahmi, Buginese, Buhid, Chakma, Cham, Devanagari, Dives Akuru, Dogra, Grantha, Gujarati, Gunjala Gondi, Gurmukhi, Gurung Khema, Hanunóo, Javanese, Kaithi, Kannada, Kawi, Kharoshthi, Khmer, Khojki, Khudawadi, Kirat Rai, Lao, Lepcha, Limbu, Makasar, Malayalam, Marchen, Masaram Gondi, Meetei Mayek, Modi, Myanmar, Nandinagari, New Tai Lue, Newa, Oriya, Phags-pa, Rejang, Saurashtra, Sharada, Siddham, Sinhala, Sora Sompeng, Soyombo, Sundanese, Syloti Nagri, Tagalog, Tagbanwa, Tai Le, Tai Tham, Tai Viet, Takri, Tamil, Telugu, Thai, Tibetan, Tirhuta, Tulu-Tigalari, Zanabazar Square
Logosyllabaries	Anatolian Hieroglyphs, Egyptian Hieroglyphs, Han, Khitan Small Script, Linear A, Nüshu, Sumero-Akkadian, Tangut
Simple Syllabaries	Bamum, Bopomofo, Canadian Aboriginal Syllabics, Cherokee, Cypriot, Cypro-Minoan, Ethiopic, Hiragana, Katakana, Linear B, Mende Kikakui, Miao, Vai, Yi
Featural Syllabaries	Hangul

Notational Systems. In addition to scripts for written natural languages, there are notational systems for other kinds of information. Some of these more closely resemble text than others. The Unicode Standard encodes symbols for use with mathematical notation, Western and Byzantine musical notation, Duployan shorthand, Sutton SignWriting notation for sign languages, and Braille, as well as symbols for use in divination, such as the Yijing hexagrams. Notational systems can be classified by how closely they resemble text. Even notational systems that do not fully resemble text may have symbols used in text. In the case of musical notation, for example, while the full notation is two-dimensional, many of the encoded symbols are frequently referenced in texts about music and musical notation.

6.2 General Punctuation

Punctuation characters—for example, U+002C COMMA and U+2022 BULLET—are encoded only once, rather than being encoded again and again for particular scripts; such general-purpose punctuation may be used for any script or mixture of scripts. In contrast, punctuation principally used with a specific script is found in the block corresponding to that script, such as U+058A ARMENIAN HYPHEN, U+061B "؛" ARABIC SEMICOLON, or the punctuation used with CJK ideographs in the CJK Symbols and Punctuation block. Script-specific punctuation characters may be unique in function, have different directionality, or be distinct in appearance or usage from their generic counterparts.

Punctuation intended for use with several related scripts is often encoded with the principal script for the group. For example, U+1735 PHILIPPINE SINGLE PUNCTUATION is encoded in a single location in the Hanunóo block, but it is intended for use with all four of the Philippine scripts.

Use and Interpretation. The use and interpretation of punctuation characters can be heavily context dependent. For example, U+002E FULL STOP can be used as sentence-ending punctuation, an abbreviation indicator, a decimal point, and so on.

Many Unicode algorithms, such as the Bidirectional Algorithm and Line Breaking Algorithm, both of which treat numeric punctuation differently from text punctuation, resolve the status of any ambiguous punctuation mark depending on whether it is part of a number context.

Legacy character encoding standards commonly include generic characters for punctuation instead of the more precisely specified characters used in printing. Examples include the single and double quotes, period, dash, and space. The Unicode Standard includes these generic characters, but also encodes the unambiguous characters independently: various forms of quotation marks, em dash, en dash, minus, hyphen, em space, en space, hair space, zero width space, and so on.

Rendering. Punctuation characters vary in appearance with the font style, just like the surrounding text characters. In some cases, where used in the context of a particular script, a specific glyph style is preferred. For example, U+002E FULL STOP should appear square when used with Armenian, but is typically circular when used with Latin. For mixed Latin/Armenian text, two fonts (or one font allowing for context-dependent glyph variation) may need to be used to render the character faithfully.

Writing Direction. Punctuation characters shared across scripts have no inherent directionality. In a bidirectional context, their display direction is resolved according to the rules in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Certain script-specific punctuation marks have an inherent directionality that matches the writing direction of the script. For an example, see "Dandas" later in this section. The image of certain paired punctuation marks, specifically those that are brackets, is mirrored when the character is part of a right-to-left directional run (see *Section 4.7, Bidi Mirrored*). Mirroring ensures that the opening and closing semantics of the character remains independent of the writing direction. The same is generally not true for other punctuation marks even when their image is not bilaterally symmetric, such as *slash* or the *curly quotes*. See also "Paired Punctuation" later in this section.

In vertical writing, many punctuation characters have special vertical glyphs. Normally, fonts contain both the horizontal and vertical glyphs, and the selection of the appropriate glyph is based on the text orientation in effect at rendering time. However, see "CJK Compatibility Forms: Vertical Forms" later in this section.

Figure 6-2 shows a set of three common shapes used for *ideographic comma* and *ideographic full stop*. The first shape in each row is that used for horizontal text, the last shape is that for vertical text. The centered form may be used with both horizontal and vertical text. See also *Figure 6-4* for an example of vertical and horizontal forms for quotation marks.

Figure 6-2. Forms of CJK Punctuation

Horizontal	Centered	Vertical

Layout Controls. A number of characters in the blocks described in this section are not graphic punctuation characters, but rather affect the operation of layout algorithms. For a description of those characters, see *Section 23.2, Layout Controls*.

Encoding Characters with Multiple Semantic Values. Some of the punctuation characters in the ASCII graphic character range (U+0020..U+007F) have multiple uses, either through ambiguity in the original standards or through accumulated reinterpretations of a limited code set. For example, 27_{16} is defined in ANSI X3.4 as *apostrophe* (*closing single quotation mark; acute accent*), and $2D_{16}$ is defined as *hyphen-minus*. In general, the Unicode Standard provides the same interpretation for the equivalent code points, without adding to or subtracting from their semantics. The Unicode Standard supplies *unambiguous* codes elsewhere for the most useful particular interpretations of these ASCII values; the corresponding unambiguous characters are cross-referenced in the character names list for this block. For more information, see "Apostrophes," "Space Characters," and "Dashes and Hyphens" later in this section.

6.2.1 Blocks Devoted to Punctuation

For compatibility with widely used legacy character sets, the Basic Latin (ASCII) block (U+0000..U+007F) and the Latin-1 Supplement block (U+0080..U+00FF) contain several of the most common punctuation signs. They are isolated from the larger body of Unicode punctuation, signs, and symbols only because their relative code locations within ASCII and Latin-1 are so widely used in standards and software. The Unicode Standard has a number of blocks devoted specifically to encoding collections of punctuation characters.

The General Punctuation block (U+2000..U+206F) contains the most common punctuation characters widely used in Latin typography, as well as a few specialized punctuation marks and a large number of format control characters. All of these punctuation characters are intended for generic use, and in principle they could be used with any script.

The Supplemental Punctuation block (U+2E00..U+2E7F) is devoted to less commonly encountered punctuation marks, including those used in specialized notational systems or occurring primarily in ancient manuscript traditions.

The CJK Symbols and Punctuation block (U+3000..U+303F) has the most commonly occurring punctuation specific to East Asian typography—that is, typography involving the rendering of text with CJK ideographs.

The Vertical Forms block (U+FE10..U+FE1F), the CJK Compatibility Forms block (U+FE30..U+FE4F), the Small Form Variants block (U+FE50..U+FE6F), and the Halfwidth and Fullwidth Forms block (U+FF00..U+FFEF) contain many compatibility characters for punctuation marks, encoded for compatibility with a number of East Asian character encoding standards. Their primary use is for round-trip mapping with those legacy standards. For vertical text, the regular punctuation characters are used instead, with alternate glyphs for vertical layout supplied by the font.

The punctuation characters in these various blocks are discussed below in terms of their general types.

6.2.2 Format Control Characters

Format control characters are special characters that have no visible glyph of their own, but that affect the display of characters to which they are adjacent, or that have other specialized functions such as serving as invisible anchor points in text. All format control characters have General_Category = Cf. A significant number of format control characters are encoded in the General Punctuation block, but their descriptions are found in other sections.

Cursive joining controls, as well as U+200B ZERO WIDTH SPACE, U+2028 LINE SEPA-RATOR, U+2029 PARAGRAPH SEPARATOR, and U+2060 WORD JOINER, are described in *Section 23.2, Layout Controls*. Bidirectional ordering controls are also discussed in *Section 23.2, Layout Controls*, but their detailed use is specified in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Invisible operators are explained in *Section 22.6, Invisible Mathematical Operators*. Deprecated format characters related to obsolete models of Arabic text processing are described in *Section 23.3, Deprecated Format Characters*.

The reserved code points U+2065 and U+FFF0..U+FFF8, as well as any reserved code points in the range U+E0000..U+E0FFF, are reserved for the possible future encoding of other format control characters. Because of this, they are treated as default ignorable code points. For more information, see *Section 5.21, Ignoring Characters in Processing*.

6.2.3 Space Characters

Space characters are found in several blocks in the Unicode Standard. The list of space characters appears in *Table 6-2*.

Table 6-2. Unicode Space Characters

Code	Name
U+0020	SPACE
U+00A0	NO-BREAK SPACE
U+1680	OGHAM SPACE MARK
U+2000	EN QUAD
U+2001	EM QUAD
U+2002	EN SPACE
U+2003	EM SPACE
U+2004	THREE-PER-EM SPACE
U+2005	FOUR-PER-EM SPACE
U+2006	SIX-PER-EM SPACE
U+2007	FIGURE SPACE
U+2008	PUNCTUATION SPACE
U+2009	THIN SPACE
U+200A	HAIR SPACE
U+202F	NARROW NO-BREAK SPACE
U+205F	MEDIUM MATHEMATICAL SPACE
U+3000	IDEOGRAPHIC SPACE

The space characters in the Unicode Standard can be identified by their General Category, (gc = Zs), in the Unicode Character Database. One exceptional "space" character is U+200B ZERO WIDTH SPACE. This character, although called a "space" in its name, does not actually have any width or visible glyph in display. It functions primarily to indicate word boundaries in writing systems that do not actually use orthographic spaces to separate words in text. It is given the General Category (gc = Cf) and is treated as a format control character, rather than as a space character, in implementations. Further discussion of U+200B ZERO WIDTH SPACE, as well as other zero-width characters with special properties, can be found in *Section 23.2, Layout Controls*.

The most commonly used space character is U+0020 SPACE. In ideographic text, U+3000 IDEOGRAPHIC SPACE is commonly used because its width matches that of the ideographs.

The main difference among other space characters is their width. U+2000..U+2006 are standard quad widths used in typography. U+2007 FIGURE SPACE has a fixed width, known as *tabular width*, which is the same width as digits used in tables. U+2008 PUNCTUATION SPACE is a space defined to be the same width as a period. U+2009 THIN SPACE and U+200A HAIR SPACE are successively smaller-width spaces used for narrow word gaps and for justification of type. The fixed-width space characters (U+2000..U+200A) are derived from conventional (hot lead) typography. Algorithmic kerning and justification in computerized typography do not use these characters. However, where they are used (for example, in typesetting mathematical formulae), their width is generally font-specified, and they typically do not expand during justification. The exception is U+2009 THIN SPACE, which sometimes gets adjusted.

In addition to the various fixed-width space characters, there are a few script-specific space characters in the Unicode Standard. U+1680 OGHAM SPACE MARK is unusual in that it is generally rendered with a visible horizontal line, rather than being blank.

No-Break Space. U+00A0 NO-BREAK SPACE (NBSP) is the nonbreaking counterpart of U+0020 SPACE. It has the same width, but behaves differently for line breaking. For more information, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm."

Unlike U+0020, U+00A0 NO-BREAK SPACE behaves as a numeric separator for the purposes of bidirectional layout. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," for a detailed discussion of the Unicode Bidirectional Algorithm.

U+00A0 NO-BREAK SPACE has an additional, important function in the Unicode Standard. It may serve as the base character for displaying a nonspacing combining mark in apparent isolation. Versions of the standard prior to Version 4.1 indicated that U+0020 SPACE could also be used for this function, but SPACE is no longer

recommended, because of potential interactions with the handling of SPACE in XML and other markup languages. See *Section 2.11, Combining Characters*, for further discussion.

Narrow No-Break Space. U+202F NARROW NO-BREAK SPACE (NNBSP) is a narrow version of U+00A0 NO-BREAK SPACE. The NNBSP can be used to represent the narrow space occurring around punctuation characters in French typography, which is called an "espace fine insécable."

Prior to Unicode Version 16.0, NNBSP was recommended as a special format character in Mongolian text. That role has been taken over by U+180E MONGOLIAN VOWEL SEPARATOR. See *Section 13.5, Mongolian*, for more information.

6.2.4 Dashes and Hyphens

Because of its prevalence in legacy encodings, U+002D HYPHEN-MINUS is the most common of the dash characters used to represent a hyphen. It has ambiguous semantic value and is rendered with an average width. U+2010 HYPHEN represents the hyphen as found in words such as "left-to-right." It is rendered with a narrow width. When typesetting text, U+2010 HYPHEN is preferred over U+002D HYPHEN-MINUS. U+2011 NON-BREAKING HYPHEN has the same semantic value as U+2010 HYPHEN, but should not be broken across lines.

U+2012 FIGURE DASH has the same (ambiguous) semantic as the U+002D HYPHEN-MINUS, but has the same width as digits (if they are monospaced). U+2013 EN DASH is used to indicate a range of values, such as 1973–1984, although in some languages *hyphen* is used for that purpose. The *en dash* should be distinguished from the U+2212 MINUS SIGN, which is an arithmetic operator. Although it is not preferred in mathematical typesetting, typographers sometimes use U+2013 EN DASH to represent the *minus sign*, particularly a *unary minus*. When interpreting formulas, U+002D HYPHEN-MINUS, U+2012 FIGURE DASH, and U+2212 MINUS SIGN should each be taken as indicating a *minus sign*, as in "x = a - b", unless a higher-level protocol precisely defines which of these characters serves that function.

U+2014 EM DASH is used to make a break—like this—in the flow of a sentence. (Some typographers prefer to use U+2013 EN DASH set off with spaces – like this – to make the same kind of break.) Like many other conventions for punctuation characters, such usage may depend on language. This kind of dash is commonly represented with a typewriter as a double hyphen. In older mathematical typography, U+2014 EM DASH may also used to indicate a *binary minus sign*. U+2015 HORIZONTAL BAR is used to introduce quoted text in some typographic styles.

U+2E3A TWO-EM DASH and U+2E3B THREE-EM DASH can be used to represent dashes even wider than an em dash. An extra-wide dash in contemporary Chinese typography, referred to as 破折號 (pòzhéhào), is used to indicate an abrupt change of

thought, the insertion of new content, or the continuation of tone or sound. That wide dash is often represented with a sequence of two U+2014 EM DASH characters, but modern practice is transitioning to the use of U+2E3A TWO-EM DASH, instead, which behaves better typographically than the sequence of em dashes. Because U+2E3A and U+2E3B are so wide, the code charts use the dashed box convention for their representative glyphs; these are, however, just ordinary punctuation characters, and not format control characters.

Dashes and hyphen characters may also be found in other blocks in the Unicode Standard. The full list is provided in *Table 6-3*. That list is correlated with the Dash property in the Unicode Character Database. Characters with the Dash property consist of all characters with General_Category=Pd plus U+2053 SWUNG DASH, U+2212 MINUS SIGN, and characters with compatibility decompositions to U+2212 MINUS SIGN.

Table 6-3. Unicode Dash Characters

Code	Name
U+002D	HYPHEN-MINUS
U+058A	ARMENIAN HYPHEN
U+05BE	HEBREW PUNCTUATION MAQAF
U+1400	CANADIAN SYLLABICS HYPHEN
U+1806	MONGOLIAN TODO SOFT HYPHEN
U+2010	HYPHEN
U+2011	NON-BREAKING HYPHEN
U+2012	FIGURE DASH
U+2013	EN DASH
U+2014	EM DASH
U+2015	HORIZONTAL BAR (= *quotation dash*)
U+2053	SWUNG DASH
U+207B	SUPERSCRIPT MINUS
U+208B	SUBSCRIPT MINUS
U+2212	MINUS SIGN
U+2E17	DOUBLE OBLIQUE HYPHEN
U+2E1A	HYPHEN WITH DIAERESIS
U+2E3A	TWO-EM DASH
U+2E3B	THREE-EM DASH
U+2E40	DOUBLE HYPHEN
U+2E5D	OBLIQUE HYPHEN
U+301C	WAVE DASH
U+3030	WAVY DASH

U+30A0	KATAKANA-HIRAGANA DOUBLE HYPHEN
U+FE31	PRESENTATION FORM FOR VERTICAL EM DASH
U+FE32	PRESENTATION FORM FOR VERTICAL EN DASH
U+FE58	SMALL EM DASH
U+FE63	SMALL HYPHEN-MINUS
U+FF0D	FULLWIDTH HYPHEN-MINUS
U+10EAD	YEZIDI HYPHENATION MARK

For a description of the line breaking behavior of dashes and hyphens, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm."

Soft Hyphen. Despite its name, U+00AD SOFT HYPHEN is not a hyphen, but rather an invisible format character used to indicate optional intraword breaks. As described in *Section 23.2, Layout Controls*, its effect on the appearance of the text depends on the language and script used.

Tilde. Although several shapes are commonly used to render U+007E "~" TILDE, modern fonts generally render it with a center line glyph, as shown here and in the code charts. However, it may also appear as a raised, spacing tilde, serving as a spacing clone of U+0303 "◌̃" COMBINING TILDE (see "Spacing Clones of Diacritical Marks" in *Section 7.9, Combining Marks*). This is a form common in older implementations, particularly for terminal emulation and typewriter-style fonts.

Some of the common uses of a tilde include indication of alternation, an approximate value, or, in some notational systems, indication of a logical negation. In the latter context, it is really being used as a shape-based substitute character for the more precise U+00AC "¬" NOT SIGN. A tilde is also used in dictionaries to repeat the defined term in examples. In that usage, as well as when used as punctuation to indicate alternation, it is more appropriately represented by a wider form, encoded as U+2053 "⁓" SWUNG DASH. U+02DC "˜" SMALL TILDE is a modifier letter encoded explicitly as the spacing form of the combining tilde as a diacritic. For mathematical usage, U+223C "∼" TILDE OPERATOR should be used to unambiguously encode the operator.

Dictionary Abbreviation Symbols. In addition to the widespread use of tilde in dictionaries, more specialized dictionaries may make use of symbols consisting of hyphens or tildes with dots or circles above or below them to abbreviate the representation of inflected or derived forms (plurals, case forms, and so on) in lexical entries. U+2E1A HYPHEN WITH DIAERESIS, for example, is typically used in German dictionaries as a short way of indicating that the addition of a plural suffix also causes placement of an umlaut on the main stem vowel. U+2E1B TILDE WITH RING ABOVE indicates a change in capitalization for a derived form, and so on. Such conventions are particularly widespread in German dictionaries, but may also appear in other dictionaries influenced by German lexicography.

6.2.5 Paired Punctuation

Mirroring of Paired Punctuation. Paired punctuation marks such as parentheses (U+0028, U+0029), square brackets (U+005B, U+005D), and braces (U+007B, U+007D) are interpreted semantically rather than graphically in the context of bidirectional or vertical texts; that is, the orientation of these characters toward the enclosed text is maintained by the software, independent of the writing direction. In a bidirectional context, the glyphs are adjusted as described in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm." (See also *Section 4.7, Bidi Mirrored*.) During display, the software must ensure that the rendered glyph is the correct one in the context of bidirectional or vertical texts.

Paired punctuation marks containing the qualifier "LEFT" in their name are taken to denote *opening*; characters whose name contains the qualifier "RIGHT" are taken to denote *closing*. For example, U+0028 LEFT PARENTHESIS and U+0029 RIGHT PARENTHESIS are interpreted as opening and closing parentheses, respectively. In a right-to-left directional run, U+0028 is rendered as ")". In a left-to-right run, the same character is rendered as "(". In some mathematical usage, brackets may not be paired, or may be deliberately used in the reversed sense, such as]a,b[. Mirroring assures that in a right-to-left environment, such specialized mathematical text continues to read]b,a[and not [b, a]. See also "Language-Based Usage of Quotation Marks" later in this section.

Quotation Marks and Brackets. Like brackets, quotation marks occur in pairs, with some overlap in usage and semantics between these two types of punctuation marks. For example, some of the CJK quotation marks resemble brackets in appearance, and they are often used when brackets would be used in non-CJK text. Similarly, both single and double *guillemets* may be treated more like brackets than quotation marks. Unlike brackets, quotation marks are not mirrored in a bidirectional context.

Some of the editing marks used in annotated editions of scholarly texts exhibit features of both quotation marks and brackets. The particular convention employed by the editors determines whether editing marks are used in pairs, which editing marks form a pair, and which is the opening character.

Horizontal brackets—for example, those used in annotating mathematical expressions—are not paired punctuation, even though the set includes both top and bottom brackets. See "Horizontal Brackets" in *Section 22.7, Technical Symbols*, for more information.

6.2.6 Language-Based Usage of Quotation Marks

The use of quotation marks differs systematically by language and by medium. As for many other punctuation marks, and in contrast to parentheses or bracket characters, quotation marks in the Unicode Standard are encoded by shape and not by how they

are used in relation to the quoted text. The same character may be used to open a quote in one language, to close a quote in another, or to serve both functions in a third.

The most commonly used character for quotation mark is U+0022 " QUOTATION MARK, usually represented with a straight double quote glyph. This quotation mark is supported on most keyboard layouts, but typographically it is mainly appropriate for typewritten manuscripts, programming text, or similar usage. Editing implementations commonly offer a facility for converting the U+0022 QUOTATION MARK to a typographically appropriate glyph for the language and context, such as distinguishing between opening or closing a quote. Within each of the different standard typographical shapes for quotation marks, there exist common glyph variations depending on font design, as discussed in the following text.

The same issues apply to the character supported on keyboards for the single quote, U+0027 APOSTROPHE; however, as its name indicates, that character is further conflated with its function as an apostrophe. (See the discussion in "Apostrophes" in this section.)

European Usage. In European typography, it is common to use *guillemets* (single or double angle quotation marks) for books and, except for some languages, curly quotation marks in office automation. Single guillemets may be used for quotes inside quotes. The following description does not attempt to be complete, but intends to document a range of known usages of quotation mark characters. Some of these usages are also illustrated in *Figure 6-3*. In this section, the words *single* and *double* are omitted from character names where there is no conflict or both are meant.

Dutch, English, Italian, Portuguese, Spanish, and Turkish use a *left quotation mark* and a *right quotation mark* for opening and closing quotations, respectively. It is typical to alternate single and double quotes for quotes within quotes. Whether single or double quotes are used for the outer quotes depends on local and stylistic conventions.

Czech, German, and Slovak use the low-9 style of quotation mark for opening instead of the standard open quotes. They employ the *left quotation mark* style of quotation mark for closing instead of the more common *right quotation mark* forms. When guillemets are used in German books, they point to the quoted text. This style is the inverse of French usage.

Danish, Finnish, Norwegian, and Swedish use the same *right quotation mark* character for both the opening and closing quotation character. This usage is employed both for office automation purposes and for books. Swedish books sometimes use the guillemet, U+00BB RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK, for both opening and closing.

Hungarian and Polish usage of quotation marks is similar to the Scandinavian usage, except that they use low double quotes for opening quotations. Presumably, these languages avoid the low single quote so as to prevent confusion with the comma.

French, Greek, Russian, and Slovenian, among others, use the guillemets, but Slovenian usage is the same as German usage in their direction. Of these languages, at least French inserts space between text and quotation marks. In the French case, U+00A0 NO-BREAK SPACE can be used for the space that is enclosed between quotation mark and text; this choice helps line breaking algorithms.

Figure 6-3. European Quotation Marks

Single right quote = apostrophe

‘quote’   don't

Usage depends on language

"English"   « French »

„German"   »Slovenian«

"Swedish"   »Swedish books»

Glyph Variation in Curly Quotes. The glyphs for the quotation marks in the range U+2018..U+201F may vary significantly across fonts. The two most typical styles use curly or wedge-shaped glyphs. See *Table 6-4*.

Table 6-4. Models of Visual Relationship between Quote Glyphs

Model	2018	2019	201A	201B	201C	201D	201E	201F
Rotated model (curly glyph style)	‘	’	‚	‛	"	"	„	‟
Rotated model (wedge glyph style)	◂	▸	▸	◂	″	″	″	″
Mirrored model (Tahoma, Verdana)	◂	▸	▸	◂	″	″	″	″

The Unicode code charts use a curly style in a serifed, Times-like font. Because quotation marks are used in pairs, glyphs within a single style are expected to be in a certain visual relationship, and that relationship stands regardless of glyph style. The visual relationship follows either a rotated or a mirrored model. The rotated model is predominant in both curly and wedge glyph style fonts. These two models are illus-

trated in *Table 6-4* using sample fonts with different glyph styles. The glyphs are enlarged for clarity.

In the rotated model, turning the ink of the glyph for U+201D RIGHT DOUBLE QUOTATION MARK 180 degrees results in the glyph for U+201C LEFT DOUBLE QUOTATION MARK; flipping it horizontally results in the glyph for U+201F DOUBLE HIGH-REVERSED-9 QUOTATION MARK. The same symmetries apply to the raised single quotation marks. Similarly, the glyphs for the low double quotation marks, U+201E DOUBLE LOW-9 QUOTATION MARK and U+2E42 DOUBLE LOW-REVERSED-9 QUOTATION MARK, are horizontally flipped images of each other.

Some fonts in widespread use instead follow the mirrored model, in which the glyph for U+201C looks like a mirrored image of the glyph for U+201D instead of a rotated image of it. Most fonts that follow the mirrored model use wedge style glyphs for quotation marks. In particular, in fonts such as Tahoma and Verdana, the glyph for U+201F is a rotated image of the glyph for U+201D, which makes the glyphs for U+201C and U+201F appear swapped compared to the typical design of wedge style quote glyphs using the rotated model. The sets of glyphs which show these swapped appearances are highlighted by a light background in *Table 6-4*.

East Asian Usage. The glyph for each quotation mark character for an Asian character set occupies predominantly a single quadrant of the character cell. The quadrant used depends on whether the character is opening or closing and whether the glyph is for use with horizontal or vertical text.

The pairs of quotation characters are listed in *Table 6-5*.

Table 6-5. East Asian Quotation Marks

Style	Opening	Closing
Corner bracket	300C	300D
White corner bracket	300E	300F
Double prime	301D	301F

Glyph Variation in East Asian Usage. In East Asian usage, the glyphs for "double-prime" quotation marks U+301D REVERSED DOUBLE PRIME QUOTATION MARK and U+301F LOW DOUBLE PRIME QUOTATION MARK consist of a pair of wedges, slanted either forward or backward, with the tips of the wedges pointing either up or down. In a pair of double-prime quotes, the closing and the opening character of the pair slant in opposite directions. Two common variations exist, as shown in *Figure 6-4*. To confuse matters more, another form of double-prime quotation marks is used with Western-style horizontal text, in addition to the curly single or double quotes.

Figure 6-4. Asian Quotation Marks

Horizontal and vertical   宇內   glyphs

『宇內』

Glyphs for overloaded character codes

〝宇內〞   "Text"

Font style-based glyph alternates

〟宇內〝   〝宇內〟
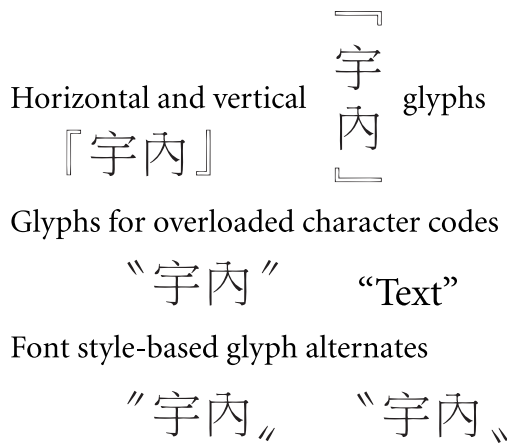
Three pairs of quotation marks are used with Western-style horizontal text, as shown in *Table 6-6*.

Table 6-6. Opening and Closing Forms

Style	Opening	Closing	Comment
Single	2018	2019	Rendered as "wide" character
Double	201C	201D	Rendered as "wide" character
Double prime	301D	301E	

Overloaded Character Codes. The character codes for standard quotes can refer to regular narrow quotes from a Latin font used with Latin text as well as to wide quotes from an Asian font used with other wide characters. This situation can be handled with some success where the text is marked up with language tags. For more information on narrow and wide characters, see Unicode Standard Annex #11, "East Asian Width."

Consequences for Semantics. The semantics of U+00AB LEFT-POINTING DOUBLE ANGLE QUOTATION MARK, U+00BB RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK, and U+201D RIGHT DOUBLE QUOTATION MARK are context dependent. By contrast, the semantics of U+201A SINGLE LOW-9 QUOTATION MARK and U+201B SINGLE HIGH-REVERSED-9 QUOTATION MARK are always opening. That usage is distinct from that of U+301F LOW DOUBLE PRIME QUOTATION MARK, which is unambiguously closing. All other quotation marks may represent opening or closing quotation marks depending on the usage.

6.2.7 Apostrophes

U+0027 ᴀᴘᴏꜱᴛʀᴏᴘʜᴇ is the most commonly used character for apostrophe. For historical reasons, U+0027 is a particularly overloaded character. In ASCII, it is used to represent a punctuation mark (such as right single quotation mark, left single quotation mark, apostrophe punctuation, vertical line, or prime) or a modifier letter (such as apostrophe modifier or acute accent). Punctuation marks generally break words; modifier letters generally are considered part of a word.

When text is set, U+2019 ʀɪɢʜᴛ ꜱɪɴɢʟᴇ ǫᴜᴏᴛᴀᴛɪᴏɴ ᴍᴀʀᴋ is preferred as apostrophe, but only U+0027 is present on most keyboards. Software commonly offers a facility for automatically converting the U+0027 ᴀᴘᴏꜱᴛʀᴏᴘʜᴇ to a contextually selected curly quotation glyph. In these systems, a U+0027 in the data stream is always represented as a straight vertical line and can never represent a curly apostrophe or a right quotation mark.

Letter Apostrophe. U+02BC ᴍᴏᴅɪꜰɪᴇʀ ʟᴇᴛᴛᴇʀ ᴀᴘᴏꜱᴛʀᴏᴘʜᴇ is preferred where the apostrophe is to represent a modifier letter (for example, in transliterations to indicate a glottal stop). In the latter case, it is also referred to as a *letter apostrophe*.

Punctuation Apostrophe. U+2019 ʀɪɢʜᴛ ꜱɪɴɢʟᴇ ǫᴜᴏᴛᴀᴛɪᴏɴ ᴍᴀʀᴋ is preferred where the character is to represent a punctuation mark, as for contractions: "*We've been here before.*" In this latter case, U+2019 is also referred to as a *punctuation apostrophe*.

An implementation cannot assume that users' text always adheres to the distinction between these characters. The text may come from different sources, including mapping from other character sets that do not make this distinction between the letter apostrophe and the punctuation apostrophe/right single quotation mark. In that case, *all* of them will generally be represented by U+2019.

The semantics of U+2019 are therefore context dependent. For example, if surrounded by letters or digits on both sides, it behaves as an in-text punctuation character and does not separate words or lines.

6.2.8 Hyphenation Point and Dictionary Syllabification

This subsection discusses several punctuation characters typically used in dictionaries to indicate syllabification and hyphenation, with a number of illustrative examples primarily from English-language dictionaries.

U+2027 ʜʏᴘʜᴇɴᴀᴛɪᴏɴ ᴘᴏɪɴᴛ is a raised dot used to indicate correct word breaking, as in "dic·tio·nary." It is a punctuation mark, to be distinguished from U+00B7 ᴍɪᴅᴅʟᴇ ᴅᴏᴛ, which has multiple semantics.

Table 6-7 gives examples of syllabification conventions in a number of English dictionaries from the 19th century to modern usage. These illustrate the usage of U+2027 HYPHENATION POINT in modern conventions, but also show the use of dashes and the interaction of syllabification punctuation marks with various modifier letters indicating syllabic stress.

Table 6-7. Dictionary Syllabification Conventions

Example	Source	Notes
SYʹLLABLE	*Dictionary of the English Language* (Samuel Johnson, 1843)	This early dictionary uses U+02B9 MODIFIER LETTER PRIME to indicate syllabic stress, placing it after the vowel letter of a syllable, rather than at the end of a syllable.
si·lă′bl	*Oxford English Dictionary* (1st Edition)	The middle dot indicates the vowel of the stressed syllable, similar to Johnson's use of a prime. The break between unstressed syllables is indicated with an apostrophe.
ˈsɪləb(ə)l	*Oxford English Dictionary* (2nd Edition)	The 2nd Edition has shifted to IPA, and indicates syllabic stress with a pre-posed U+02C8 MODIFIER LETTER VERTICAL LINE. The parentheses indicate optional omission of the *schwa* in pronunciation.
sil′ə-bl	*Chambers English Dictionary* (7th Edition)	This dictionary shows syllabic breaks, where the stressed syllable is followed by U+02B9 MODIFIER LETTER PRIME and the dash is U+2010 HYPHEN, to indicate a preferred hyphenation location. When splitting a word like **abate′-ment**, the stress mark goes after the stressed syllable, followed directly by the hyphen.
sɪ̲ləbl	*BBC English Dictionary*	This dictionary uses IPA without indication of syllabic breaks in the pronunciation. The vowel of a stressed syllable is underscored, hence ɪ̲ can be represented as <U+026A, U+0332>.
sɪ̲ləbəᵒl	*Collins Cobuild English Language Dictionary*	This convention is similar to the *BBC English Dictionary*. The raised zero is represented by U+2070 ⁰ SUPERSCRIPT ZERO and indicates the optional omission of the *schwa*.
syl·la·ble (**sílləb′l**)	*Readers Digest Great Illustrated Dictionary*	This exemplifies a modern convention of separating representation of syllabification and of pronunciation. The syllabification for word break is indicated by the use of U+2027 in the spelling. The primary stress is indicated with an accent on the vowel, instead of a modifier letter, and the apostrophe indicates the omitted *schwa*.
syl·la·ble /ˈsiləbəl/	*Webster's 3rd New International Dictionary*	This edition of Webster's uses broad IPA for the pronunciation, shown between phonemic slashes. It splits words at the end of a line with a normal hyphen. When a word is split at the end of a line at the position of a dash in the spelling, that is indicated in the dictionary by the use of U+2E17 ⸗ DOUBLE OBLIQUE HYPHEN.

sy·lla·ble /ˈsɪl.ə.bəl/ [ˈsɪl.ə.bɫ̩]	*Wiktionary*	Online dictionaries construct their entries quite differently than traditional printed dictionaries, and typically separate hyphenation from pronunciation. Wiktionary often omits hyphenation, but when listed it uses the hyphenation point to show hyphenation opportunities. It may separately list a broad IPA pronunciation and a close IPA pronunciation of the word. In the IPA pronunciation, a U+002E FULL STOP indicates phonological syllable boundaries.

Some dictionaries use a character that looks like a vertical series of four dots to indicate places where there is a syllable, but no allowable break, as for example, **a⦙plomb** or **hoar⦙y**. This convention can be represented by U+205E VERTICAL FOUR DOTS. To hint that this punctuation mark should not itself be an opportunity for a line break, it can be followed by U+2060 WORD JOINER.

See also the discussion of dictionary abbreviation symbols involving hyphens and tildes in *Section 6.2.4, Dashes and Hyphens*.

Interaction with Line Breaking. Where possible, the line breaking properties for punctuation marks commonly used in dictionaries have been assigned so as to accommodate these and similar conventions by default. However, implementing the full conventions in dictionaries requires tailoring of line break classes and rules or other types of special support. See Unicode Standard Annex #14, *Unicode Line Breaking Algorithm*, for the specification of line break classes and how to tailor them.

6.2.9 Other Punctuation

Tironian Et. U+204A TIRONIAN SIGN ET acts as a punctuation mark meaning "and". It can also function as a letter in some contexts. In some Medieval language materials, Tironian *et* can appear in uppercase and is represented by U+2E52 TIRONIAN SIGN CAPITAL ET. However, TIRONIAN SIGN ET and TIRONIAN SIGN CAPITAL ET are not case-mapped.

Word Separator Middle Dot. Historic texts in many scripts, especially those that are handwritten (manuscripts), sometimes use a raised dot to separate words. Such word-separating punctuation is comparable in function to the use of space to separate words in modern typography.

U+2E31 WORD SEPARATOR MIDDLE DOT is a middle dot punctuation mark which is analogous in function to the script-specific character U+16EB RUNIC SINGLE PUNCTUATION, but is for use with any script that needs a raised dot for separating words. For example, it can be used for the word-separating dot seen in Avestan or Samaritan texts.

Fraction Slash. U+2044 FRACTION SLASH is used between digits to form numeric fractions, such as 2/3 and 3/9. The standard form of a fraction built using the fraction

slash is defined as follows: any sequence of one or more decimal digits (General Category = Nd), followed by the fraction slash, followed by any sequence of one or more decimal digits. Such a fraction should be displayed as a unit, such as ¾ or $\frac{3}{4}$. The precise choice of display can depend on additional formatting information.

If the displaying software is incapable of mapping the fraction to a unit, then it can also be displayed as a simple linear sequence as a fallback (for example, 3/4). If the fraction is to be separated from a previous number, then a space can be used, choosing the appropriate width (normal, thin, zero width, and so on). For example, 1 + THIN SPACE + 3 + FRACTION SLASH + 4 is displayed as 1¾.

Spacing Overscores and Underscores. U+203E OVERLINE is the above-the-line counterpart to U+005F LOW LINE. It is a spacing character, not to be confused with U+0305 COMBINING OVERLINE. As with all overscores and underscores, a sequence of these characters should connect in an unbroken line. The overscoring characters also must be distinguished from U+0304 COMBINING MACRON, which does not connect horizontally in this way.

Doubled Punctuation. Several doubled punctuation characters that have compatibility decompositions into a sequence of two punctuation marks are also encoded as single characters: U+203C DOUBLE EXCLAMATION MARK, U+2048 QUESTION EXCLAMATION MARK, and U+2049 EXCLAMATION QUESTION MARK. These doubled punctuation marks are included as an implementation convenience for East Asian and Mongolian text, when rendered vertically.

Period or Full Stop. The *period*, or U+002E FULL STOP, can be circular or square in appearance, depending on the font or script. The hollow circle period used in East Asian texts is separately encoded as U+3002 IDEOGRAPHIC FULL STOP. Likewise, Armenian, Arabic, Ethiopic, and several other script-specific periods are coded separately because of their significantly different appearance.

In contrast, the various functions of the period, such as its use as sentence-ending punctuation, an abbreviation mark, or a decimal point, are not separately encoded. The specific semantic therefore depends on context.

In old-style numerals, where numbers vary in placement above and below the baseline, a decimal or thousands separator may be displayed with a dot that is raised above the baseline. Because it would be inadvisable to have a stylistic variation between old-style and new-style numerals that actually changes the underlying representation of text, the Unicode Standard considers this raised dot to be merely a glyphic variant of U+002E "." FULL STOP.

Ellipsis. The omission of text is often indicated by a sequence of three dots "...", a punctuation convention called *ellipsis*. Typographic traditions vary in how they lay out these dots. In some cases the dots are closely spaced; in other cases the dots are spaced farther apart. U+2026 "…" HORIZONTAL ELLIPSIS is the ordinary Unicode

character intended for the representation of an ellipsis in text and typically shows the dots separated with a moderate degree of spacing. A sequence of three U+002E FULL STOP characters can also be used to indicate an ellipsis, in which case the space between the dots will depend on the font used for rendering.

In a monowidth font, a sequence of three *full stops* will be wider than *horizontal ellipsis*, and may be appropriate when following style guides that require more widely spaced dots. In this case, the spacing between the last dot and following punctuation would be as expected.

In contrast, for a typical proportional font, a *full stop* is very narrow and a sequence of three of them will be more tightly spaced than the dots in *horizontal ellipsis*. When adhering to style guides, such as the Chicago Manual of Style (CMOS), which call for more widely spaced dots in presentation, established practice calls for separating the dots (and any surrounding punctuation) by either a NBSP or NNBSP. These contrasts are illustrated in *Table 6-8*, using "dots" to refer to *full stop* characters, with or without intervening no break spaces.

<div align="center">

Table 6-8. Horizontal Ellipsis

</div>

	Ellipsis	**dots**	**dots + NBSP**	**dots + NNBSP**
Fixed	`abc…def`	`abc...def`	`abc . . . def`	`abc . . . def`
	`abc.…Def`	`abc....Def`	`abc. . . . Def`	`abc. . . . Def`
Proportional	`abc…def`	`abc...def`	`abc . . . def`	`abc...def`
	`abc.…Def`	`abc....Def`	`abc. . . . Def`	`abc.... Def`

There are conventions that use four dots for an ellipsis in certain grammatical contexts, such as elided content following the end of a sentence. These conventions can represent the four dots either as a sequence of <full stop, horizontal ellipsis> or <horizontal ellipsis, full stop> or simply as a sequence of four *full stop* characters, depending on the requirements of those conventions. The usual CMOS convention of placing a *full stop* initially in these sequences is also illustrated in *Table 6-8*.

In East Asian typographic traditions, particularly in Japan, an ellipsis is raised to the center line of text. When an ellipsis is represented by U+2026 HORIZONTAL ELLIPSIS or by sequences of full stops, this effect requires specialized rendering support. In practice, it is relatively common for authors of East Asian text to substitute U+22EF MIDLINE HORIZONTAL ELLIPSIS for this. Because the midline ellipsis is a mathematical symbol, intended to represent column elision in matrix notation, it is typically used with layout on a mathematical center line. With appropriate font design to harmonize with East Asian typography, this midline ellipsis can produce the desired appearance without having to support contextual shifting of the baseline for U+2026 HORIZONTAL ELLIPSIS.

Vertical Ellipsis. When text is laid out vertically, the ellipsis is normally oriented so that the dots run from top to bottom. Most commonly, an East Asian font will contain a vertically oriented glyph variant of U+2026 for use in vertical text layout. U+FE19 PRESENTATION FORM FOR VERTICAL HORIZONTAL ELLIPSIS is a compatibility character for use in mapping to the GB 18030 standard; it would not usually be used for an ellipsis except in systems that cannot handle the contextual choice of glyph variants for vertical rendering.

U+22EE VERTICAL ELLIPSIS and U+22EF MIDLINE HORIZONTAL ELLIPSIS are part of a set of special ellipsis characters used for row or column elision in matrix notation. Although their primary use is for a mathematical context, U+22EF MIDLINE HORIZONTAL ELLIPSIS has also become popular for the midline ellipsis in East Asian typography. When U+22EF is used this way, an East Asian font will typically contain a rotated glyph variant for use in vertical text layout. If an appropriate mechanism for glyph variant substitution (such as the "vert" GSUB feature in the Open Font Format) in vertically rendered text is not available, U+FE19 PRESENTATION FORM FOR VERTICAL HORIZONTAL ELLIPSIS is the preferred character substitution to represent a vertical ellipsis, instead of the mathematical U+22EE VERTICAL ELLIPSIS.

U+205D TRICOLON has a superficial resemblance to a *vertical ellipsis*, but is part of a set of dot delimiter punctuation marks for various manuscript traditions. As for the *colon*, the dots in the *tricolon* are always oriented vertically.

Leader Dots. Leader dots are typically seen in contexts such as a table of contents or in indices, where they represent a kind of style line, guiding the eye from an entry in the table to its associated page number. Usually leader dots are generated automatically by page formatting software and do not require the use of encoded characters. However, there are occasional plain text contexts in which a string of leader dots is represented as a sequence of characters. U+2024 ONE DOT LEADER and U+2025 TWO DOT LEADER are intended for such usage. U+2026 HORIZONTAL ELLIPSIS can also serve as a three-dot version of leader dots. These leader dot characters can be used to control, to a certain extent, the spacing of leader dots based on font design, in contexts where a simple sequence of *full stops* will not suffice.

U+2024 ONE DOT LEADER also serves as a "semicolon" punctuation in Armenian, where it is distinguished from U+002E FULL STOP. See *Section 7.6, Armenian*.

Other Basic Latin Punctuation Marks. The interword punctuation marks encoded in the Basic Latin block are used for a variety of other purposes. This can complicate the tasks of parsers trying to determine sentence boundaries. As noted later in this section, some can be used as numeric separators. Both *period* and U+003A ":" COLON can be used to mark abbreviations as in "etc." or as in the Swedish abbreviation "S:ta" for "Sankta". U+0021 "!" EXCLAMATION MARK is used as a mathematical operator (*factorial*). U+003F "?" QUESTION MARK is often used as a substitution character when mapping Unicode characters to other character sets where they do not have a

representation. This practice can lead to unexpected results when the converted data are file names from a file system that supports "?" as a wildcard character.

Several punctuation marks, such as *colon*, *middle dot* and *solidus* closely resemble mathematical operators, such as U+2236 RATIO, U+22C5 DOT OPERATOR and U+2215 DIVISION SLASH. The latter are the preferred characters, but the former are often substituted because they are more easily typed.

Canonical Equivalence Issues for Greek Punctuation. Some commonly used Greek punctuation marks are encoded in the Greek and Coptic block, but are canonical equivalents to generic punctuation marks encoded in the C0 Controls and Basic Latin block, because they are indistinguishable in shape. Thus, U+037E ";" GREEK QUESTION MARK is canonically equivalent to U+003B ";" SEMICOLON, and U+0387 "·" GREEK ANO TELEIA is canonically equivalent to U+00B7 "·" MIDDLE DOT. In these cases, as for other canonical singletons, the preferred form is the character that the canonical singletons are mapped to, namely U+003B and U+00B7 respectively. Those are the characters that will appear in any normalized form of Unicode text, even when used in Greek text as Greek punctuation. Text segmentation algorithms need to be aware of this issue, as the kinds of text units delimited by a *semicolon* or a *middle dot* in Greek text will typically differ from those in Latin text.

The character properties for U+00B7 MIDDLE DOT are particularly problematical, in part because of identifier issues for that character. There is no guarantee that all of its properties align exactly with U+0387 GREEK ANO TELEIA, because the latter's properties are based on the limited function of the *middle dot* in Greek as a delimiting punctuation mark.

Bullets. U+2022 BULLET is the typical character for a bullet. Within the general punctuation, several alternative forms for bullets are separately encoded: U+2023 TRIANGULAR BULLET, U+204C BLACK LEFTWARDS BULLET, and so on. U+00B7 MIDDLE DOT also often functions as a small bullet. Bullets mark the head of specially formatted paragraphs, often occurring in lists, and may use arbitrary graphics or dingbat forms as well as more conventional bullet forms. U+261E WHITE RIGHT POINTING INDEX, for example, is often used to highlight a note in text, as a kind of gaudy bullet.

Paragraph Marks. U+00A7 SECTION SIGN and U+00B6 PILCROW SIGN are often used as visible indications of sections or paragraphs of text, in editorial markup, to show format modes, and so on. Which character indicates sections and which character indicates paragraphs may vary by convention. U+204B REVERSED PILCROW SIGN is a fairly common alternate representation of the paragraph mark.

Numeric Separators. Any of the characters U+002C COMMA, U+002E FULL STOP, and the Arabic characters U+060C, U+066B, or U+066C (and possibly others) can

be used as numeric separator characters, depending on the locale and user customizations.

Obelus. Originally a punctuation mark to denote questionable passages in manuscripts, U+00F7 ÷ DIVISION SIGN is now most commonly used as a symbol indicating division. However, modern use is not limited to that meaning. The character is sometimes used to indicate a range (similar to the *en-dash*) or as a form of *minus sign*. The former use is attested for Russian, Polish and Italian, and latter use is still widespread in Scandinavian countries in some contexts, but may occur elsewhere as well. (See also the following text on "Commercial Minus.")

Commercial Minus. U+2052 ⁒ COMMERCIAL MINUS SIGN is used in commercial or tax-related forms or publications in several European countries, including Germany and Scandinavia. The string "./." is used as a fallback representation for this character.

The symbol may also appear as a marginal note in letters, denoting enclosures. One variation replaces the top dot with a digit indicating the number of enclosures.

An additional usage of the sign appears in the Uralic Phonetic Alphabet (UPA), where it marks a structurally related borrowed element of different pronunciation. In Finland and a number of other European countries, the dingbats ⁒ and ✓ are always used for "correct" and "incorrect," respectively, in marking a student's paper. This contrasts with American practice, for example, where ✓ and ✗ might be used for "correct" and "incorrect," respectively, in the same context.

At Sign. U+0040 COMMERCIAL AT has acquired a prominent modern use as part of the syntax for e-mail addresses. As a result, users in practically every language community suddenly needed to use and refer to this character. Consequently, many colorful names have been invented for this character. Some of these contain references to animals or even pastries. *Table 6-9* gives a sample.

Table 6-9. Names for the @

Language	Name and Comments
Chinese	= xiǎo lǎoshǔ (means "little mouse" in Mandarin Chinese) = lǎoshǔ hào (means "mouse mark" in Mandarin Chinese)
Danish	= snabel-a
Dutch	= apenstaartje (common, humorous slang)
Finnish	= ät, ät-merkki (Finnish standard) = kissanhäntä, miukumauku (common, humorous slang)
French	= arobase, arrobe, escargot, a crolle (common, humorous slang)
German	= Klammeraffe
Hebrew	= shtrudl ("Strudel", modern Hebrew) = krukhit (more formal Hebrew)

Hungarian	= kukac (common, humorous slang)
Italian	= chiocciola
Polish	= atka, małpa, małpka (common, humorous slang)
Portuguese	= arroba
Russian	= sobachka (common, humorous slang)
Slovenian	= afna (common, humorous slang)
Spanish	= arroba
Swedish	= snabel-a, kanelbulle (common, humorous slang)

6.2.10 Archaic Punctuation and Editorial Marks

Archaic Punctuation. Many archaic scripts use punctuation marks consisting of a set of multiple dots, such as U+2056 THREE DOT PUNCTUATION. The semantics of these marks can vary by script, and some of them are also used for special conventions, such as the use of U+205E VERTICAL FOUR DOTS in modern dictionaries. U+205B FOUR DOT MARK and U+205C DOTTED CROSS were used by scribes in the margin to highlight a piece of text. More of these multiple-dot archaic punctuation marks are encoded in the range U+2E2A..U+2E2D.

These kinds of punctuation marks occur in ancient scripts and are also common in medieval manuscripts. Their specific functions may be different in each script or manuscript tradition. However, encoding only a single set in the Unicode Standard simplifies the task of deciding which character to use for a given mark.

There are some exceptions to this general rule. Archaic scripts with script-specific punctuation include Runic, Aegean Numbers, and Cuneiform. In particular, the appearance of punctuation written in the Cuneiform style is sufficiently different that no unification was attempted.

Double Oblique Hyphen. U+2E17 "⸗" DOUBLE OBLIQUE HYPHEN is used in ancient Near Eastern linguistics to indicate certain morphological boundaries while continuing to use the ordinary hyphen to indicate other boundaries. This symbol is also semantically distinct from U+003D "=" EQUALS SIGN. Fraktur fonts use an oblique glyph of similar appearance for the hyphen, but that is merely a font variation of U+002D HYPHEN-MINUS or U+2010 HYPHEN, not the distinctly encoded DOUBLE OBLIQUE HYPHEN.

Editorial Marks. In addition to common-use editorial marks such as U+2041 CARET INSERTION POINT encoded in the General Punctuation block, there are a number of editorial marks encoded in the Supplemental Punctuation block (U+2E00..U+2E7F). Editorial marks differ from ordinary punctuation marks, in that their primary purpose is to allow editors to mark up a scholarly publication of a text to show the location and contents of insertions, omissions, variant readings, and other such information about the text.

The half brackets encoded in the range U+2E22..U+2E25 are widely used as editorial marks in critical editions of ancient and medieval texts. They appear, for example, in editions of transliterated Cuneiform and ancient Egyptian texts. U+2E26 LEFT SIDE-WAYS U BRACKET and U+2E27 RIGHT SIDEWAYS U BRACKET are a specialized bracket pair used in some traditions, and should be distinguished from mathematical set symbols of similar appearance. The double parentheses are employed by Latinists.

New Testament Editorial Marks. The Greek text of the New Testament exists in a large number of manuscripts with many textual variants. The most widely used critical edition of the New Testament, the Nestle-Aland edition published by the United Bible Societies (UBS), introduced a set of editorial characters that are regularly used in a number of journals and other publications. As a result, these editorial marks have become the recognized method of annotating the New Testament.

U+2E00 RIGHT ANGLE SUBSTITUTION MARKER is placed at the start of a single word when that word is replaced by one or more different words in some manuscripts. These alternative readings are given in the *apparatus criticus*. If there is a second alternative reading in one verse, U+2E01 RIGHT ANGLE DOTTED SUBSTITUTION MARKER is used instead.

U+2E02 LEFT SUBSTITUTION BRACKET is placed at the start of a sequence of words where an alternative reading is given in the *apparatus criticus*. This bracket is used together with the U+2E03 RIGHT SUBSTITUTION BRACKET. If there is a second alternative reading in one verse, the dotted forms at U+2E04 and U+2E05 are used instead.

U+2E06 RAISED INTERPOLATION MARKER is placed at a point in the text where another version has additional text. This additional text is given in the *apparatus criticus*. If there is a second piece of interpolated text in one verse, the dotted form U+2E07 RAISED DOTTED INTERPOLATION MARKER is used instead.

U+2E08 DOTTED TRANSPOSITION MARKER is placed at the start of a word or verse that has been transposed. The transposition is explained in the *apparatus criticus*. When the words are preserved in different order in some manuscripts, U+2E09 LEFT TRANSPOSITION BRACKET is used. The end of such a sequence of words is marked by U+2E0A RIGHT TRANSPOSITION BRACKET.

The characters U+2E0B RAISED SQUARE and U+2E0C LEFT RAISED OMISSION BRACKET are conventionally used in pairs to bracket text, with RAISED SQUARE marking the start of a passage of omitted text and LEFT RAISED OMISSION BRACKET marking its end. In other editorial traditions, U+2E0C LEFT RAISED OMISSION BRACKET may be paired with U+2E0D RIGHT RAISED OMISSION BRACKET. Depending on the conventions used, either may act as the starting or ending bracket.

Two other bracket characters, U+2E1C LEFT LOW PARAPHRASE BRACKET and U+2E1D RIGHT LOW PARAPHRASE BRACKET, have particular usage in the N'Ko script, but also may be used for general editorial punctuation.

Ancient Greek Editorial Marks. Ancient Greek scribes generally wrote in continuous uppercase letters without separating letters into words. On occasion, the scribe added punctuation to indicate the end of a sentence or a change of speaker or to separate words. Editorial and punctuation characters appear abundantly in surviving papyri and have been rendered in modern typography when possible, often exhibiting considerable glyphic variation. A number of these editorial marks are encoded in the range U+2E0E..U+2E16.

The punctuation used in Greek manuscripts can be divided into two categories: marginal or semi-marginal characters that mark the end of a section of text (for example, *coronis*, *paragraphos*), and characters that are mixed in with the text to mark pauses, end of sense, or separation between words (for example, *stigme*, *hypodiastole*). The *hypodiastole* is used in contrast with *comma* and is not a glyph variant of it.

A number of editorial characters are attributed to and named after Aristarchos of Samothrace (circa 216–144 BCE), fifth head of the Library at Alexandria. Aristarchos provided a major edition of the works of Homer, which forms the basis for modern editions.

A variety of Ancient Greek editorial marks are shown in the text of *Figure 6-5*, including the *editorial coronis* and *upwards ancora* on the left. On the right are illustrated the *dotted obelos*, *capital dotted lunate sigma symbol*, *capital reversed lunate sigma symbol*, and a glyph variant of the *downards ancora*. The numbers on the left indicate text lines. A *paragraphos* appears below the start of line 12. The opening brackets "[" indicate fragments, where text is illegible or missing in the original. These examples are slightly adapted and embellished from editions of the *Oxyrhynchus Papyri* and Homer's *Iliad*.

Figure 6-5. Examples of Ancient Greek Editorial Marks

U+2E0F PARAGRAPHOS is placed at the beginning of the line but may refer to a break in the text at any point in the line. The *paragraphos* should be a horizontal line, generally stretching under the first few letters of the line it refers to, and possibly extending into the margin. It should be given a no-space line of its own and does not itself constitute a line or paragraph break point for the rest of the text. Examples of the *paragraphos*, *forked paragraphos*,and *reversed forked paragraphos* are illustrated in *Figure 6-6*.

Figure 6-6. Use of Greek Paragraphos

δαιμονα...　　　　δαιμονα...　　　　δαιμονα...
δευοντοσου...　　　δευοντοσου...　　δευοντοσου...

6.2.11 Indic Punctuation

Dandas. Dandas are phrase-ending punctuation common to the scripts of South and South East Asia. The Devanagari *danda* and *double danda* characters are intended for generic use across the scripts of India. They are also occasionally used in Latin transliteration of traditional texts from Indic scripts.

There are minor visual differences in the appearance of the dandas, which may require script-specific fonts or a font that can provide glyph alternates based on script environment. For the four Philippine scripts, the analogues to the dandas are encoded once in Hanunóo and shared across all four scripts. The other Brahmi-derived scripts have separately encoded equivalents for the danda and double danda. In some scripts, as for Tibetan, multiple, differently ornamented versions of dandas may occur. The dandas encoded in the Unicode Standard are listed in *Table 6-10*.

Table 6-10. Unicode Danda Characters

Code	Name
U+0964	DEVANAGARI DANDA
U+0965	DEVANAGARI DOUBLE DANDA
U+0E5A	THAI CHARACTER ANGKHANKHU
U+0F08	TIBETAN MARK SBRUL SHAD
U+0F0D	TIBETAN MARK SHAD
U+0F0E	TIBETAN MARK NYIS SHAD
U+0F0F	TIBETAN MARK TSHEG SHAD
U+0F10	TIBETAN MARK NYIS TSHEG SHAD
U+0F11	TIBETAN MARK RIN CHEN SPUNGS SHAD
U+0F12	TIBETAN MARK RGYA GRAM SHAD
U+104A	MYANMAR SIGN LITTLE SECTION

U+104B	MYANMAR SIGN SECTION
U+1735	PHILIPPINE SINGLE PUNCTUATION
U+1736	PHILIPPINE DOUBLE PUNCTUATION
U+17D4	KHMER SIGN KHAN
U+17D5	KHMER SIGN BARIYOOSAN
U+1AA8	TAI THAM SIGN KAAN
U+1AA9	TAI THAM SIGN KAANKUU
U+1B5E	BALINESE CARIK SIKI
U+1B5F	BALINESE CARIK PAREREN
U+1C3B	LEPCHA PUNCTUATION TA-ROL
U+1C3C	LEPCHA PUNCTUATION NYET THYOOM TA-ROL
U+1C7E	OL CHIKI PUNCTUATION MUCAAD
U+1C7F	OL CHIKI PUNCTUATION DOUBLE MUCAAD
U+A876	PHAGS-PA MARK SHAD
U+A877	PHAGS-PA MARK DOUBLE SHAD
U+A8CE	SAURASHTRA DANDA
U+A8CF	SAURASHTRA DOUBLE DANDA
U+A92F	KAYAH LI SIGN SHYA
U+A9C8	JAVANESE PADA LINGSA
U+A9C9	JAVANESE PADA LUNGSI
U+AA5D	CHAM PUNCTUATION DANDA
U+AA5E	CHAM PUNCTUATION DOUBLE DANDA
U+AA5F	CHAM PUNCTUATION TRIPLE DANDA
U+AAF0	MEETEI MAYEK CHEIKHAN
U+ABEB	MEETEI MAYEK CHEIKHEI
U+10A56	KHAROSHTHI PUNCTUATION DANDA
U+10A57	KHAROSHTHI PUNCTUATION DOUBLE DANDA
U+11047	BRAHMI DANDA
U+11048	BRAHMI DOUBLE DANDA
U+110C0	KAITHI DANDA
U+110C1	KAITHI DOUBLE DANDA
U+11141	CHAKMA DANDA
U+11142	CHAKMA DOUBLE DANDA
U+111C5	SHARADA DANDA
U+111C6	SHARADA DOUBLE DANDA
U+11238	KHOJKI DANDA
U+11239	KHOJKI DOUBLE DANDA
U+113D4	TULU-TIGALARI DANDA
U+113D5	TULU-TIGALARI DOUBLE DANDA

U+1144B	NEWA DANDA
U+1144C	NEWA DOUBLE DANDA
U+115C2	SIDDHAM DANDA
U+115C3	SIDDHAM DOUBLE DANDA
U+11641	MODI DANDA
U+11642	MODI DOUBLE DANDA
U+1173C	AHOM SIGN SMALL SECTION
U+1173D	AHOM SIGN SECTION
U+11944	DIVES AKURU DOUBLE DANDA
U+11A42	ZANABAZAR SQUARE MARK SHAD
U+11A43	ZANABAZAR SQUARE MARK DOUBLE SHAD
U+11A9B	SOYOMBO MARK SHAD
U+11A9C	SOYOMBO MARK DOUBLE SHAD
U+11C41	BHAIKSUKI DANDA
U+11C42	BHAIKSUKI DOUBLE DANDA
U+11C71	MARCHEN MARK SHAD
U+11F43	KAWI DANDA
U+11F44	KAWI DOUBLE DANDA
U+16A6E	MRO DANDA
U+16A6F	MRO DOUBLE DANDA
U+16D6E	KIRAT RAI DANDA
U+16D6F	KIRAT RAI DOUBLE DANDA

The Bidirectional Class of the dandas matches that for the scripts they are intended for. Kharoshthi, which is written from right to left, has Bidirectional Class R for U+10A56 KHAROSHTHI PUNCTUATION DANDA. For more on bidirectional classes, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Note that the name of the danda in Hindi is *viram*, while the different Unicode character named *virama* is called *halant* in Hindi. If this distinction is not kept in mind, it can lead to confusion as to which character is meant.

6.2.12 CJK Punctuation

CJK Punctuation comprises punctuation marks and symbols used by writing systems that employ Han ideographs. Most of these characters are found in East Asian standards. Typical for many of these wide punctuation characters is that the actual image occupies only the left or the right half of the normal square character cell. The extra whitespace is frequently removed in a kerning step during layout, as shown in *Figure 6-7*. Unlike ordinary kerning, which uses tables supplied by the font, the char-

acter space adjustment of wide punctuation characters is based on their character code.

Figure 6-7. CJK Parentheses

FF08   FF08   After Kerning
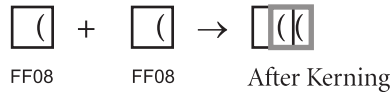
U+3000 IDEOGRAPHIC SPACE is provided for compatibility with legacy character sets. It is a fixed-width wide space appropriate for use with an ideographic font. For more information about wide characters, see Unicode Standard Annex #11, "East Asian Width."

U+3030 WAVY DASH is a special form of a dash found in East Asian character standards. (For a list of other space and dash characters in the Unicode Standard, see *Table 6-2* and *Table 6-3*.)

U+3037 IDEOGRAPHIC TELEGRAPH LINE FEED SEPARATOR SYMBOL is a visible indicator of the line feed separator symbol used in the Chinese telegraphic code. It is comparable to the pictures of control codes found in the Control Pictures block.

U+3005 IDEOGRAPHIC ITERATION MARK is used to stand for the second of a pair of identical ideographs occurring in adjacent positions within a document.

U+3006 IDEOGRAPHIC CLOSING MARK is used frequently on signs to indicate that a store or booth is closed for business. The Japanese pronunciation is *shime*, most often encountered in the compound *shime-kiri*.

The U+3008 and U+3009 angle brackets are unambiguously wide, as are other bracket characters in this block, such as double angle brackets, tortoise shell brackets, and white square brackets. Where mathematical and other non-CJK contexts use brackets of similar shape, the Unicode Standard encodes them separately.

U+3012 POSTAL MARK is used in Japanese addresses immediately preceding the numerical postal code. It is also used on forms and applications to indicate the blank space in which a postal code is to be entered. U+3020 POSTAL MARK FACE and U+3036 CIRCLED POSTAL MARK are properly glyphic variants of U+3012 and are included for compatibility.

U+3031 VERTICAL KANA REPEAT MARK and U+3032 VERTICAL KANA REPEAT WITH VOICED SOUND MARK are used only in *vertically written* Japanese to repeat pairs of kana characters occurring immediately prior in a document. The voiced variety U+3032 is used in cases where the repeated kana are to be voiced. For instance, a repetitive phrase like *toki-doki* could be expressed as <U+3068, U+304D, U+3032> in vertical writing. Both of these characters are intended to be represented by "double-height" glyphs requiring two ideographic "cells" to print; this intention also

explains the existence in source standards of the characters representing the top and bottom halves of these characters (that is, the characters U+3033, U+3034, and U+3035). In horizontal writing, similar characters are used, and they are separately encoded. In Hiragana, the equivalent repeat marks are encoded at U+309D and U+309E; in Katakana, they are U+30FD and U+30FE.

Wave Dash. U+301C WAVE DASH is a compatibility character that was originally encoded to represent the character in the JIS C 6226-1978 standard and all subsequent revisions and extensions with the *kuten* code: 1-33 (0x8160 in Shift-JIS encoding). The mapping of this character has been problematical. Some major implementations originally mapped, and continue to map for compatibility purposes, that JIS character to U+FF5E FULLWIDTH TILDE, instead. The mapping issue has been documented in the Unicode Standard since Version 3.0.

From Version 2.0 through Version 7.0 of the Unicode Standard, U+301C was shown in the code charts with a representative glyph that had a wide *reversed* tilde shape. Starting with Version 8.0, however, the representative glyph has been corrected to a wide tilde shape, to reflect predominant practice in commercial fonts. For most purposes, U+301C WAVE DASH should be treated simply as a duplicate representation of U+FF5E FULLWIDTH TILDE.

Sesame Dots. U+FE45 SESAME DOT and U+FE46 WHITE SESAME DOT are used in vertical text, where a series of sesame dots may appear beside the main text, as a sidelining to provide visual emphasis. In this respect, their usage is similar to such characters as U+FE34 PRESENTATION FORM FOR VERTICAL WAVY LOW LINE, which are also used for sidelining vertical text for emphasis. Despite being encoded in the block for CJK compatibility forms, the sesame dots are not compatibility characters. They are in general typographic use and are found in the Japanese standard, JIS X 0213.

U+FE45 SESAME DOT is historically related to U+3001 IDEOGRAPHIC COMMA, but is not simply a vertical form variant of it. The function of an *ideographic comma* in connected text is distinct from that of a *sesame dot*.

6.2.13 Unknown or Unavailable Ideographs

U+3013 GETA MARK is used to indicate the presence of, or to hold a place for, an ideograph that is not available when a document is printed. It has no other use. Its name comes from its resemblance to the mark left by traditional Japanese sandals (*geta*). A variety of light and heavy glyphic variants occur.

U+303E IDEOGRAPHIC VARIATION INDICATOR is a graphic character that is to be rendered visibly. It alerts the user that the intended character is similar to, but not equal to, the character that follows. Its use is similar to the existing character U+3013 GETA MARK. A GETA MARK substitutes for the unknown or unavailable character,

but does not identify it. The IDEOGRAPHIC VARIATION INDICATOR is the head of a two-character sequence that gives some indication about the intended glyph or intended character. Ultimately, the IDEOGRAPHIC VARIATION INDICATOR and the character following it are intended to be replaced by the correct character, once it has been identified or a font resource or input resource has been provided for it.

U+303F IDEOGRAPHIC HALF FILL SPACE is a visible indicator of a display cell filler used when ideographic characters have been split during display on systems using a double-byte character encoding. It is included in the Unicode Standard for compatibility.

See also "Ideographic Description Sequences" in *Section 18.2, Ideographic Description Characters*.

6.2.14 CJK Compatibility Forms

Vertical Forms. CJK vertical forms are compatibility characters encoded for compatibility with legacy implementations that encode these characters explicitly when Chinese text is being set in vertical rather than horizontal lines. The preferred Unicode approach to representation of such text is to simply use the nominal characters that correspond to these vertical variants. Then, at display time, the appropriate glyph is selected according to the line orientation.

The Unicode Standard contains two blocks devoted primarily to these CJK vertical forms. The CJK Vertical Forms block, U+FE10..U+FE1F, contains compatibility characters needed for round-trip mapping to the Chinese standard, GB 18030. The CJK Compatibility Forms block, U+FE30..U+FE4F, contains forms found in the Chinese standard, CNS 11643.

Styled Overscores and Underscores. The CJK Compatibility Forms block also contains a number of compatibility characters from CNS 11643, which consist of different styles of overscores or underscores. They were intended, in the Chinese standard, for the representation of various types of overlining or underlining, for emphasis of text when laid out *horizontally*. Except for round-trip mapping with legacy character encodings, the use of these characters is discouraged; use of styles is the preferred way to handle such effects in modern text rendering.

Small Form Variants. CNS 11643 also contains a number of small variants of ASCII punctuation characters. The Unicode Standard encodes those variants as compatibility characters in the Small Form Variants block, U+FE50..U+FE6F. Those characters, while construed as fullwidth characters, are nevertheless depicted using small forms that are set in a fullwidth display cell. (See the discussion in *Section 18.4, Hiragana and Katakana*.) These characters are provided for compatibility with legacy implementations.

Two small form variants from CNS 11643/plane 1 were unified with other characters outside the ASCII block: 2131_{16} was unified with U+00B7 MIDDLE DOT, and 2261_{16} was unified with U+2215 DIVISION SLASH.

Fullwidth and Halfwidth Variants. For compatibility with East Asian legacy character sets, the Unicode Standard encodes fullwidth variants of ASCII punctuation and halfwidth variants of CJK punctuation. See *Section 18.5, Halfwidth and Fullwidth Forms*, for more information.

Chapter 7

Europe-I
Modern and Liturgical Scripts

Modern European alphabetic scripts are derived from or influenced by the Greek script, which itself was an adaptation of the Phoenician alphabet. A Greek innovation was writing the letters from left to right, which is the writing direction for all the scripts derived from or inspired by Greek.

Some scripts whose geographic area of primary usage is outside Europe are included in this chapter because of their relationship with Greek script. Coptic is used primarily by the Coptic church in Egypt and elsewhere; Armenian and Georgian are primarily associated with countries in the Caucasus (which is often not included as part of Europe), although Armenian in particular is used by a large diaspora.

These scripts are all written from left to right. Many have separate lowercase and uppercase forms of the alphabet. Spaces are used to separate words. Accents and diacritical marks are used to indicate phonetic features and to extend the use of base scripts to additional languages. Some of these modification marks have evolved into small free-standing signs that can be treated as characters in their own right.

The Latin script is used to write or transliterate texts in a wide variety of languages. The International Phonetic Alphabet (IPA) is an extension of the Latin alphabet, enabling it to represent the phonetics of all languages. Other Latin phonetic extensions are used for the Uralic Phonetic Alphabet and the Teuthonista transcription system.

The Latin alphabet is derived from the alphabet used by the Etruscans, who had adopted a Western variant of the classical Greek alphabet (*Section 8.6, Old Italic*). Originally it contained only 24 capital letters. The modern Latin alphabet as it is found in the Basic Latin block owes its appearance to innovations of scribes during the Middle Ages and practices of the early Renaissance printers.

The Cyrillic script was developed in the ninth century and is also based on Greek. Like Latin, Cyrillic is used to write or transliterate texts in many languages. The Georgian and Armenian scripts were devised in the fifth century and are influenced by Greek.

The Coptic script was the last stage in the development of Egyptian writing. It represented the adaptation of the Greek alphabet to writing Egyptian, with the retention of forms from Demotic for sounds not adequately represented by Greek letters. Although primarily used in Egypt from the fourth to the tenth century, it is described in this chapter because of its close relationship to the Greek script.

Glagolitic is an early Slavic script related in some ways to both the Greek and the Cyrillic scripts. It was widely used in the Balkans but gradually died out, surviving the longest in Croatia. Like Coptic, however, it still has some modern use in liturgical contexts.

This chapter also describes modifier letters and combining marks used with the Latin script and other scripts.

The block descriptions for other archaic European alphabetic scripts, such as Gothic, Ogham, Old Italic, and Runic, can be found in *Chapter 8, Europe-II*.

7.1 Latin

The Latin script was derived from the Greek script. Today it is used to write a wide variety of languages all over the world. In the process of adapting it to other languages, numerous extensions have been devised. The most common is the addition of diacritical marks. Furthermore, the creation of digraphs, inverse or reverse forms, and outright new characters have all been used to extend the Latin script.

The Latin script is written in linear sequence from left to right. Spaces are used to separate words and provide the primary line breaking opportunities. Hyphens are used where lines are broken in the middle of a word. (For more information, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm.") Latin letters come in uppercase and lowercase pairs.

Languages. Some indication of language or other usage is given for many characters within the names lists accompanying the character charts.

Diacritical Marks. Speakers of different languages treat the addition of a diacritical mark to a base letter differently. In some languages, the combination is treated as a letter in the alphabet for the language. In others, such as English, the same words can often be spelled with and without the diacritical mark without implying any difference. Most languages that use the Latin script treat letters with diacritical marks as variations of the base letter, but do not accord the combination the full status of an independent letter in the alphabet. Widely used accented character combinations are provided as single characters to accommodate interoperation with pervasive practice in legacy encodings. Combining diacritical marks can express these and all other accented letters as combining character sequences.

In the Unicode Standard, all diacritical marks are encoded in sequence *after the base characters to which they apply*. For more details, see the subsection "Combining Diacritical Marks" in *Section 7.9, Combining Marks*, and also *Section 2.11, Combining Characters*.

Alternative Glyphs. Some characters have alternative representations, although they have a common semantic. In such cases, a preferred glyph is chosen to represent the character in the code charts, even though it may not be the form used under all circumstances. Some Latin examples to illustrate this point are provided in *Figure 7-1* and discussed in the text that follows.

Figure 7-1. Alternative Glyphs in Latin

① a ɑ   g ɡ

② ď ď ɖ   ť ť   ľ ľ

③ ǵ ģ ģ   ④ ţ ţ
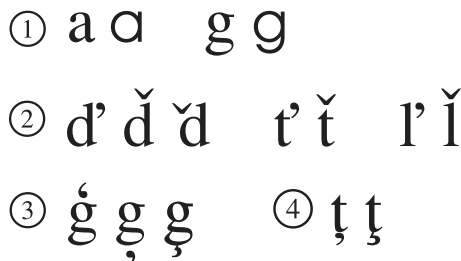
Common typographical variations of basic Latin letters include the open- and closed-loop forms of the lowercase letters "a" and "g", as shown in the first example in *Figure 7-1*. In ordinary Latin text, such distinctions are merely glyphic alternates for the same characters; however, phonetic transcription systems, such as IPA, often make systematic distinctions between these forms.

Variations in Diacritical Marks. The shape and placement of diacritical marks can be subject to considerable variation that might surprise a reader unfamiliar with such distinctions. For example, when Czech is typeset, U+010F LATIN SMALL LETTER D WITH CARON and U+0165 LATIN SMALL LETTER T WITH CARON are often rendered by glyphs with an apostrophe instead of with a caron, commonly known as a háček. See the second example in *Figure 7-1*. In Slovak, this use also applies to U+013E LATIN SMALL LETTER L WITH CARON and U+013D LATIN CAPITAL LETTER L WITH CARON. The use of an apostrophe can avoid some line crashes over the ascenders of those letters and so result in better typography. In typewritten or handwritten documents, or in didactic and pedagogical material, glyphs with háčeks are preferred.

Characters with cedillas, commas or ogoneks below often are subject to variable typographical usage, depending on the availability and quality of fonts used, the technology, the era and the geographic area. Various hooks, cedillas, commas, and squiggles may be substituted for the nominal forms of these diacritics below, and even the directions of the hooks may be reversed.

The character U+0327 COMBINING CEDILLA can be displayed by a wide variety of forms, including cedillas and commas below. This variability also occurs for the precomposed characters whose decomposition includes U+0327. For text in some languages, a specific form is typically preferred. In particular, Latvian and Romanian prefer a comma below, while a cedilla is preferred in Turkish and Marshallese. These language-specific preferences are discussed in more detail in the text that follows.

Also, as a result of legacy encodings and practices, and the mapping of those legacy encodings to Unicode, some particular shapes for U+0327 COMBINING CEDILLA are preferred in the absence of language or locale context. A rendering as cedilla is

preferred for the letters listed in the first column, while rendering as comma below is preferred for those listed in the second column of *Table 7-1*.

Table 7-1. Preferred Rendering of Cedilla versus Comma Below

Cedilla	Comma Below
c, e, h, s	d, g, k, l, n, r, t

Latvian Cedilla. There is specific variation involved in the placement and shapes of cedillas on Latvian characters. This is illustrated by the Latvian letter U+0123 LATIN SMALL LETTER G WITH CEDILLA, as shown in example 3 in *Figure 7-1*. In good Latvian typography, this character is always shown with a rotated comma *over* the g, rather than a cedilla below the g, because of the typographical design and layout issues resulting from trying to place a cedilla below the descender loop of the g. Poor Latvian fonts may substitute an acute accent for the rotated comma, and handwritten or other printed forms may actually show the cedilla below the g. The uppercase form of the letter is always shown with a cedilla, as the rounded bottom of the *G* poses no problems for attachment of the cedilla.

Other Latvian letters with a cedilla below (U+0137 LATIN SMALL LETTER K WITH CEDILLA, U+0146 LATIN SMALL LETTER N WITH CEDILLA, and U+0157 LATIN SMALL LETTER R WITH CEDILLA) always prefer a glyph with a floating comma below, as there is no proper attachment point for a cedilla at the bottom of the base form.

Cedilla and Comma Below in Turkish and Romanian. The Latin letters *s* and *t* with comma below or with cedilla diacritics pose particular interpretation issues for Turkish and Romanian data, both in legacy character sets and in the Unicode Standard. Legacy character sets generally include a single form for these characters. While the formal interpretation of legacy character sets is that they contain only one of the forms, in practice this single character has been used to represent any of the forms. For example, 0xBA in ISO 8859-2 is formally defined as a *lowercase s with cedilla*, but has been used to represent a *lowercase s with comma below* for Romanian.

The Unicode Standard provides unambiguous representations for all of the forms, for example, U+0219 ș LATIN SMALL LETTER S WITH COMMA BELOW versus U+015F ş LATIN SMALL LETTER S WITH CEDILLA. In modern usage, the preferred representation of Romanian text is with U+0219 ș LATIN SMALL LETTER S WITH COMMA BELOW, while Turkish data is represented with U+015F ş LATIN SMALL LETTER S WITH CEDILLA.

However, due to the prevalence of legacy implementations, a large amount of Romanian data will contain U+015F ş LATIN SMALL LETTER S WITH CEDILLA or the corresponding code point 0xBA in ISO 8859-2. When converting data represented using ISO 8859-2, 0xBA should be mapped to the appropriate form. When processing Romanian Unicode data, implementations should treat U+0219 ș LATIN SMALL

LETTER S WITH COMMA BELOW and U+015F ş LATIN SMALL LETTER S WITH CEDILLA as equivalent.

Exceptional Case Pairs. The characters U+0130 LATIN CAPITAL LETTER I WITH DOT ABOVE and U+0131 LATIN SMALL LETTER DOTLESS I (used primarily in Turkish) are assumed to take ASCII "i" and "I", respectively, as their case alternates. This mapping makes the corresponding reverse mapping language-specific; mapping in both directions requires special attention from the implementer (see *Section 5.18, Case Mappings*).

Diacritics on i and j. A dotted (normal) *i* or *j* followed by some common nonspacing marks above loses the dot in rendering. Thus, in the word *naïve,* the *ï* could be spelled with *i + diaeresis*. A *dotted-i* is not equivalent to a Turkish *dotless-i + overdot*, nor are other cases of accented *dotted-i* equivalent to accented *dotless-i* (for example, i + ¨ ≠ ı + ¨). The same pattern is used for *j. Dotless-j* is used in the *Landsmålsalfabet*, where it does not have a case pair.

To express the forms sometimes used in the Baltic (where the dot is retained under a top accent in dictionaries), use *i + overdot + accent* (see *Figure 7-2*).

Figure 7-2. Diacritics on *i* and *j*

All characters that use their dot in this manner have the Soft_Dotted property in Unicode.

Vietnamese. In the modern Vietnamese alphabet, there are 12 vowel letters and 5 tone marks (see *Figure 7-3*). Normalization Form C represents the combination of vowel letter and tone mark as a single unit—for example, U+1EA8 Ẩ LATIN CAPITAL LETTER A WITH CIRCUMFLEX AND HOOK ABOVE. Normalization Form D decomposes this combination into the combining character sequence, such as <U+0041, U+0302, U+0309>. Some widely used implementations prefer storing the vowel letter and the tone mark separately.

Figure 7-3. Vietnamese Letters and Tone Marks

The Vietnamese vowels and other letters are found in the Basic Latin, Latin-1 Supplement, and Latin Extended-A blocks. Additional precomposed vowels and tone marks are found in the Latin Extended Additional block.

The characters U+0300 COMBINING GRAVE ACCENT, U+0309 COMBINING HOOK ABOVE, U+0303 COMBINING TILDE, U+0301 COMBINING ACUTE ACCENT, and U+0323 COMBINING DOT BELOW should be used in representing the Vietnamese tone marks. The characters U+0340 COMBINING GRAVE TONE MARK and U+0341 COMBINING ACUTE TONE MARK have canonical equivalences to U+0300 COMBINING GRAVE ACCENT and U+0301 COMBINING ACUTE ACCENT, respectively; they are not recommended for use in representing Vietnamese tones, despite the presence of *tone mark* in their character names.

Standards. Unicode follows ISO/IEC 8859-1 in the layout of Latin letters up to U+00FF. ISO/IEC 8859-1, in turn, is based on older standards—among others, ASCII (ANSI X3.4), which is identical to ISO/IEC 646:1991-IRV. Like ASCII, ISO/IEC 8859-1 contains Latin letters, punctuation signs, and mathematical symbols. These additional characters are widely used with scripts other than Latin. The descriptions of these characters are found in *Chapter 6, Writing Systems and Punctuation*, and *Chapter 22, Symbols*.

The Latin Extended-A block includes characters contained in ISO/IEC 8859—Part 2. *Latin alphabet No. 2*, Part 3. *Latin alphabet No. 3*, Part 4. *Latin alphabet No. 4*, and Part 9. *Latin alphabet No. 5*. Many of the other graphic characters contained in these standards, such as punctuation, signs, symbols, and diacritical marks, are already encoded in the Latin-1 Supplement block. Other characters from these parts of ISO/IEC 8859 are encoded in other blocks, primarily in the Spacing Modifier Letters block (U+02B0..U+02FF) and in the character blocks starting at and following the General Punctuation block. The Latin Extended-A block also covers additional characters from ISO/IEC 6937.

The Latin Extended-B block covers, among others, characters in ISO 6438 Documentation—African coded character set for bibliographic information interchange, *Pinyin* Latin transcription characters from the People's Republic of China national standard GB 2312 and from the Japanese national standard JIS X 0212, and Sami characters from ISO/IEC 8859 Part 10. *Latin alphabet No. 6*.

The characters in the IPA block are taken from the 1989 revision of the International Phonetic Alphabet, published by the International Phonetic Association. Extensions from later IPA sources have also been added.

Related Characters. For other Latin-derived characters, see Letterlike Symbols (U+2100..U+214F), Currency Symbols (U+20A0..U+20CF), Number Forms (U+2150..U+218F), Enclosed Alphanumerics (U+2460..U+24FF), CJK Compatibility

(U+3300..U+33FF), Fullwidth Forms (U+FF21..U+FF5A), and Mathematical Alphanumeric Symbols (U+1D400..U+1D7FF).

7.1.1 Letters of Basic Latin: U+0041–U+007A

Only a small fraction of the languages written with the Latin script can be written entirely with the basic set of 26 uppercase and 26 lowercase Latin letters contained in this block. The 26 basic letter pairs form the core of the alphabets used by all the other languages that use the Latin script. A stream of text using one of these alphabets would therefore intermix characters from the Basic Latin block and other Latin blocks.

Occasionally a few of the basic letter pairs are not used to write a language. For example, Italian does not use "j" or "w".

7.1.2 Letters of the Latin-1 Supplement: U+00C0–U+00FE

The Latin-1 supplement extends the basic 26 letter pairs of ASCII by providing additional letters for the major languages of Europe listed in the next paragraph.

Languages. The languages supported by the Latin-1 supplement include Catalan, Danish, Dutch, Faroese, Finnish, Flemish, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Spanish, and Swedish.

Ordinals. U+00AA FEMININE ORDINAL INDICATOR and U+00BA MASCULINE ORDINAL INDICATOR can be depicted with an underscore, but many modern fonts show them as superscripted Latin letters with no underscore. In sorting and searching, these characters should be treated as weakly equivalent to their Latin character equivalents.

7.1.3 Latin Extended-A: U+0100–U+017F

The Latin Extended-A block contains a collection of letters that, when added to the letters contained in the Basic Latin and Latin-1 Supplement blocks, allow for the representation of most European languages that employ the Latin script. Many other languages can also be written with the characters in this block. Most of these characters are equivalent to precomposed combinations of base character forms and combining diacritical marks. These combinations may also be represented by means of composed character sequences. See *Section 2.11, Combining Characters*, and *Section 7.9, Combining Marks*.

Compatibility Digraphs. The Latin Extended-A block contains five compatibility digraphs, encoded for compatibility with ISO/IEC 6937:1984. Two of these characters, U+0140 LATIN SMALL LETTER L WITH MIDDLE DOT and its uppercase version, were originally encoded in ISO/IEC 6937 for support of Catalan. In current conventions,

the representation of this digraphic sequence in Catalan simply uses a sequence of an ordinary "l" and U+00B7 MIDDLE DOT.

Another pair of characters, U+0133 LATIN SMALL LIGATURE IJ and its uppercase version, was provided to support the digraph "ij" in Dutch, often termed a "ligature" in discussions of Dutch orthography. When adding intercharacter spacing for line justification, the "ij" is kept as a unit, and the space between the *i* and *j* does not increase. In titlecasing, both the *i* and the *j* are uppercased, as in the word "IJsselmeer." Using a single code point might simplify software support for such features; however, because a vast amount of Dutch data is encoded without this digraph character, under most circumstances one will encounter an <i, j> sequence.

Finally, U+0149 LATIN SMALL LETTER N PRECEDED BY APOSTROPHE was encoded for use in Afrikaans. The character is deprecated, and its use is strongly discouraged. In nearly all cases it is better represented by a sequence of an apostrophe followed by "n".

Languages. Most languages supported by this block also require the concurrent use of characters contained in the Basic Latin and Latin-1 Supplement blocks. When combined with these two blocks, the Latin Extended-A block supports Afrikaans, Basque, Breton, Croatian, Czech, Esperanto, Estonian, French, Frisian, Greenlandic, Hungarian, Latin, Latvian, Lithuanian, Maltese, Polish, Provençal, Rhaeto-Romanic, Romanian, Romany, Sámi, Slovak, Slovenian, Sorbian, Turkish, Welsh, and many others.

7.1.4 Latin Extended-B: U+0180–U+024F

The Latin Extended-B block contains letterforms used to extend Latin scripts to represent additional languages. It also contains phonetic symbols not included in the International Phonetic Alphabet (see the IPA Extensions block, U+0250..U+02AF).

Arrangement. The characters are arranged in a nominal alphabetical order, followed by a small collection of Latinate forms. Uppercase and lowercase pairs are placed together where possible, but in many instances the other case form is encoded at some distant location and so is cross-referenced. Variations on the same base letter are arranged in the following order: turned, inverted, hook attachment, stroke extension or modification, different style, small cap, modified basic form, ligature, and Greek derived.

Croatian Digraphs Matching Serbian Cyrillic Letters. Serbo-Croatian is a single language with paired alphabets: a Latin script (Croatian) and a Cyrillic script (Serbian). A set of compatibility digraph codes is provided for one-to-one transliteration. There are two potential uppercase forms for each digraph, depending on whether only the initial letter is to be capitalized (titlecase) or both (all uppercase). The Unicode Standard offers both forms so that software can convert one form to the

other without changing font sets. The appropriate cross references are given for the lowercase letters.

Pinyin Diacritic–Vowel Combinations. The Chinese standard GB 2312, the Japanese standard JIS X 0212, and some other standards include codes for Pinyin, which is used for Latin transcription of Mandarin Chinese. Most of the letters used in Pinyin romanization are already covered in the preceding Latin blocks. The group of 16 characters provided here completes the Pinyin character set specified in GB 2312 and JIS X 0212.

Case Pairs. A number of characters in this block are uppercase forms of characters whose lowercase forms are part of some other grouping. Many of these characters came from the International Phonetic Alphabet; they acquired uppercase forms when they were adopted into Latin script-based writing systems. Occasionally, however, *alternative* uppercase forms arose in this process. In some instances, research has shown that alternative uppercase forms are merely variants of the same character. If so, such variants are assigned a single Unicode code point, as is the case of U+01B7 LATIN CAPITAL LETTER EZH. But when research has shown that two uppercase forms are actually used in different ways, then they are given different codes; such is the case for U+018E LATIN CAPITAL LETTER REVERSED E and U+018F LATIN CAPITAL LETTER SCHWA. In this instance, the shared lowercase form is copied to enable unique case-pair mappings: U+01DD LATIN SMALL LETTER TURNED E is a copy of U+0259 LATIN SMALL LETTER SCHWA.

For historical reasons, the names of some case pairs differ. For example, U+018E LATIN CAPITAL LETTER REVERSED E is the uppercase of U+01DD LATIN SMALL LETTER TURNED E—not of U+0258 LATIN SMALL LETTER REVERSED E. For default case mappings of Unicode characters, see *Section 4.2, Case*.

Caseless Letters. A number of letters used with the Latin script are caseless—for example, the caseless *glottal stop* at U+0294 and U+01BB LATIN LETTER TWO WITH STROKE, and the various letters denoting click sounds. Caseless letters retain their shape when uppercased. When titlecasing words, they may also act transparently; that is, if they occur in the leading position, the next following cased letter may be uppercased instead.

Over the last several centuries, the trend in typographical development for the Latin script has tended to favor the eventual introduction of case pairs. See the following discussion of the glottal stop. The Unicode Standard may encode additional uppercase characters in such instances. However, for reasons of stability, the standard will never add a new lowercase form for an existing uppercase character. See also "Caseless Matching" in *Section 5.18, Case Mappings*.

Glottal Stop. There are two patterns of usage for the *glottal stop* in the Unicode Standard. U+0294 ʔ LATIN LETTER GLOTTAL STOP is a caseless letter used in IPA. It is also

widely seen in language orthographies based on IPA or Americanist phonetic usage, in those instances where no casing is apparent for *glottal stop*. Such orthographies may avoid casing for *glottal stop* to the extent that when titlecasing strings, a word with an initial *glottal stop* may have its second letter uppercased instead of the first letter.

In a small number of orthographies for languages of northwestern Canada, and in particular, for Chipewyan, Dogrib, and Slavey, case pairs have been introduced for *glottal stop*. For these orthographies, the cased *glottal stop* characters should be used: U+0241 ʔ LATIN CAPITAL LETTER GLOTTAL STOP and U+0242 ʔ LATIN SMALL LETTER GLOTTAL STOP.

The glyphs for the *glottal stop* are somewhat variable and overlap to a certain extent. The glyph shown in the code charts for U+0294 ʔ LATIN LETTER GLOTTAL STOP is a cap-height form as specified in IPA, but the same character is often shown with a glyph that resembles the top half of a question mark and that may or may not be cap height. U+0241 ʔ LATIN CAPITAL LETTER GLOTTAL STOP, while shown with a larger glyph in the code charts, often appears identical to U+0294. U+0242 ʔ LATIN SMALL LETTER GLOTTAL STOP is a small form of U+0241.

Various small, raised hook- or comma-shaped characters are often substituted for a *glottal stop*—for instance, U+02BC ʼ MODIFIER LETTER APOSTROPHE, U+02BB ʻ MODIFIER LETTER TURNED COMMA, U+02C0 ʔ MODIFIER LETTER GLOTTAL STOP, or U+02BE ʾ MODIFIER LETTER RIGHT HALF RING. U+02BB, in particular, is used in Hawaiian orthography as the *ʻokina*.

Click Letters. Historically there have been a number of conventions for writing click consonants, using either the Latin alphabet or special symbols. Three systems are notable:

- 1. The three Latin letters *c q x* used for Xhosa orthography. These were adopted to write related Bantu languages, as well as to non-Bantu languages such as Julʼhoansi, Naro, Sandawe, and Hadza, sometimes with *ç* added for palatal clicks.

- 2. The old-style IPA letters ʇ ʖ ʗ ʞ. These were extended by various authors with the additional letters ⊙ ƒ ⱷ, of which ⊙ was adopted by the IPA, but they remained in minority usage and apart from ⊙ were abandoned by the IPA in 1989.

- 3. The Lepsius letters. These originated in a simple bar [ǀ] that developed into the letters [ǀ] [ǁ] [ǃ] [ǂ] of current IPA practice, as defined by the 1999 *Handbook* and subsequent IPA chart updates. These letters are used in the orthographies of the majority of Khoisan languages.

These three conventions are summarized in . The column headers for place of articulation use the terms currently favored by linguists.

Table 7-2. Alternative Systems of Click Letters

Articulation	bilabial	dental	alveolar	lateral	palatal	retroflex
Latin (Xhosa)		c 0063	q 0071	x 0078	ç 00E7	
Extended old IPA	ʘ 0298	ʇ 0287	ʗ 0297	ʖ 0296	ƒ 1DF0B	ψ 03C8
Current IPA	ʘ 0298	ǀ 01C0	ǃ 01C3	ǁ 01C1	ǂ 01C2	ǃ̢ 1DF0A

The history of terminology for the place of articulation of clicks listed in the "alveolar" and "palatal" columns has been complicated and confusing. The names of the Unicode characters for these clicks inherited an earlier understanding of articulation. Thus U+01C2 ǂ LATIN LETTER ALVEOLAR CLICK is currently analyzed in IPA as having a palatoalveolar articulation and is referred to as the *palatal click*, whereas U+01C3 ǃ LATIN LETTER RETROFLEX CLICK is analyzed as having a (post)alveolar articulation and is referred to as the *alveolar click*. The Unicode character names are immutable identifiers and cannot be updated to track the changing terminology of articulatory phonetics.

Former Latin letters for palatal clicks are U+0076 v and U+0254 ɔ. The ad hoc symbol U+203C ‼ DOUBLE EXCLAMATION MARK has been used for retroflex clicks in what is otherwise IPA transcription. The dedicated letter U+1DF0A ǃ̢ LATIN LETTER RETROFLEX CLICK WITH RETROFLEX HOOK is "implicit" in the IPA but is not included on the <u>summary IPA chart</u>. Occasionally U+2980 ⦀ TRIPLE VERTICAL BAR DELIMITER is used for a second lateral click. The retired "velar" click letter U+029E ʞ LATIN SMALL LETTER TURNED K has seen use in the 21st century for a paralexical back-released click.

These letters take IPA diacritics or form multigraphs to indicate whether the click consonant is nasal, voiced, aspirated, glottalized, and so forth. In language orthographies, only a couple of the accompanying letters require any clarification—specifically U+0294 ʔ in the early 20th century and its modern equivalent U+02BC ʼ MODIFIER LETTER APOSTROPHE or, rarely, U+02EE ʺ MODIFIER LETTER DOUBLE APOSTROPHE.

A character occasionally used for a generic click consonant is U+A7B0 Ʞ LATIN CAPITAL LETTER TURNED K. Sometimes U+1DF10 𝼐 LATIN LETTER SMALL CAPITAL TURNED K is substituted to provide more room for combining IPA diacritics.

7.1.5 IPA Extensions: U+0250–U+02AF

The IPA Extensions block contains primarily the unique symbols of the International Phonetic Alphabet, which is a standard system for indicating specific speech sounds. The IPA was first introduced in 1886 and has undergone occasional revisions of

content and usage since that time. The Unicode Standard covers all single symbols and all diacritics in the IPA revision of 1999, as well as a few symbols in former IPA usage that are no longer currently sanctioned. A few symbols have been added to this block that are part of the transcriptional practices of Sinologists, Americanists, and other linguists. Some of these practices have usages independent of the IPA and may use characters from other Latin blocks rather than IPA forms. Other late additions to IPA, as well as a number of nonstandard or obsolete phonetic symbols are located in the Latin Extended-B or Latin Extended-C blocks.

An essential feature of IPA is the use of combining diacritical marks. IPA diacritical mark characters are coded in the Combining Diacritical Marks block, U+0300..U+036F. In IPA, diacritical marks can be freely applied to base form letters to indicate the fine degrees of phonetic differentiation required for precise recording of different languages.

Standards. The International Phonetic Association standard considers IPA to be a separate alphabet, so it includes the entire Latin lowercase alphabet *a–z*, a number of extended Latin letters such as U+0153 œ LATIN SMALL LIGATURE OE, and a few Greek letters and other symbols as separate and distinct characters. In contrast, the Unicode Standard does not duplicate either the Latin lowercase letters *a–z* or other Latin or Greek letters in encoding IPA. Unlike other character standards referenced by the Unicode Standard, IPA constitutes an extended alphabet and phonetic transcriptional standard, rather than a character encoding standard.

Unifications. The IPA characters are unified as much as possible with other letters, albeit not with nonletter symbols such as U+222B ∫ INTEGRAL. The IPA characters have also been adopted into the Latin-based alphabets of many written languages, such as some used in Africa. It is futile to attempt to distinguish a transcription from an actual alphabet in such cases. Therefore, many IPA characters are found outside the IPA Extensions block. IPA characters that are not found in the IPA Extensions block are listed as cross references at the beginning of the character names list for this block.

IPA Alternates. In a few cases IPA practice has, over time, produced alternate forms, such as U+0269 "ɩ" LATIN SMALL LETTER IOTA versus U+026A "ɪ" LATIN LETTER SMALL CAPITAL I. The Unicode Standard provides separate encodings for the two forms because they are used in a meaningfully distinct fashion.

Case Pairs. IPA does not sanction case distinctions; in effect, its phonetic symbols are all lowercase. When IPA symbols are adopted into a particular alphabet and used by a given written language (as has occurred, for example, in Africa), they acquire uppercase forms. Because these uppercase forms are not themselves IPA symbols, they are generally encoded in the Latin Extended-B block (or other Latin extension blocks) and are cross-referenced with the IPA names list.

Typographic Variants. IPA includes typographic variants of certain Latin and Greek letters that would ordinarily be considered variations of font style rather than of character identity, such as SMALL CAPITAL letterforms. Examples include a typographic variant of the Greek letter *phi* φ and the borrowed letter Greek *iota* ι, which has a unique Latin uppercase form. These forms are encoded as separate characters in the Unicode Standard because they have distinct semantics in plain text.

Affricate Digraph Ligatures. IPA officially sanctions six digraph ligatures used in transcription of coronal affricates. These are encoded at U+02A3..U+02A8. The IPA digraph ligatures are explicitly defined in IPA and have possible semantic values that make them not simply rendering forms. For example, while U+02A6 LATIN SMALL LETTER TS DIGRAPH is a transcription for the sounds that could also be transcribed in IPA as "ts" <U+0074, U+0073>, the choice of the digraph ligature may be the result of a deliberate distinction made by the transcriber regarding the systematic phonetic status of the affricate. The choice of whether to ligate cannot be left to rendering software based on the font available. This ligature also differs in typographical design from the "ts" ligature found in some old-style fonts.

Arrangement. The IPA Extensions block is arranged in approximate alphabetical order according to the Latin letter that is graphically most similar to each symbol. This order has nothing to do with a phonetic arrangement of the IPA letters.

7.1.6 Phonetic Extensions: U+1D00–U+1D7F

Most of the characters in the first of the two adjacent blocks comprising the phonetic extensions are used in the Uralic Phonetic Alphabet (UPA; also called Finno-Ugric Transcription, FUT), a highly specialized system that has been used by Uralicists globally for more than 100 years. Originally, it was chiefly used in Finland, Hungary, Estonia, Germany, Norway, Sweden, and Russia, but it is now known and used worldwide, including in North America and Japan. Uralic linguistic description, which treats the phonetics, phonology, and etymology of Uralic languages, is also used by other branches of linguistics, such as Indo-European, Turkic, and Altaic studies, as well as by other sciences, such as archaeology.

A very large body of descriptive texts, grammars, dictionaries, and chrestomathies exists, and continues to be produced, using this system.

The UPA makes use of approximately 258 characters, some of which are encoded in the Phonetic Extensions block; others are encoded in the other Latin blocks and in the Greek and Cyrillic blocks. The UPA takes full advantage of combining characters. It is not uncommon to find a base letter with three diacritics above and two below.

Typographic Features of the UPA. Small capitalization in the UPA means voicelessness of a normally voiced sound. Small capitalization is also used to indicate certain either voiceless or half-voiced consonants. Superscripting indicates very short schwa

vowels or transition vowels, or in general very short sounds. Subscripting indicates co-articulation caused by the preceding or following sound. Rotation (turned letters) indicates reduction; sideways (that is, 90 degrees counterclockwise) rotation is used where turning (180 degrees) might result in an ambiguous representation.

UPA phonetic material is generally represented with italic glyphs, so as to separate it from the surrounding text.

Other Phonetic Extensions. The remaining characters in the phonetics extension range U+1D6C..U+1DBF are derived from a wide variety of sources, including many technical orthographies developed by SIL linguists, as well as older historic sources.

All attested phonetic characters showing struckthrough tildes, struckthrough bars, and retroflex or palatal hooks attached to the basic letter have been separately encoded here. Although separate combining marks exist in the Unicode Standard for overstruck diacritics and attached retroflex or palatal hooks, earlier encoded IPA letters such as U+0268 LATIN SMALL LETTER I WITH STROKE and U+026D LATIN SMALL LETTER L WITH RETROFLEX HOOK have never been given decomposition mappings in the standard. For consistency, all newly encoded characters are handled analogously to the existing, more common characters of this type and are not given decomposition mappings. Because these characters do not have decompositions, they require special handling in some circumstances. See the discussion of single-script confusables in Unicode Technical Standard #39, "Unicode Security Mechanisms."

The Phonetic Extensions Supplement block also contains 37 superscript modifier letters. These complement the much more commonly used superscript modifier letters found in the Spacing Modifier Letters block.

U+1D77 LATIN SMALL LETTER TURNED G and U+1D78 MODIFIER LETTER CYRILLIC EN are used in Caucasian linguistics. U+1D79 LATIN SMALL LETTER INSULAR G is used in older Irish phonetic notation. It is to be distinguished from a Gaelic style glyph for U+0067 LATIN SMALL LETTER G.

Digraph for th. U+1D7A LATIN SMALL LETTER TH WITH STRIKETHROUGH is a digraphic notation commonly found in some English-language dictionaries, representing the voiceless (inter)dental fricative, as in *th*in. While this character is clearly a digraph, the obligatory strikethrough across two letters distinguishes it from a "th" digraph per se, and there is no mechanism involving combining marks that can easily be used to represent it. A common alternative glyphic form for U+1D7A uses a horizontal bar to strike through the two letters, instead of a diagonal stroke.

7.1.7 Latin Extended Additional: U+1E00–U+1EFF

The characters in this block are mostly precomposed combinations of Latin letters with one or more general diacritical marks. With the exception of U+1E9A LATIN

SMALL LETTER A WITH RIGHT HALF RING, each of the precomposed characters contained in this block is a canonical decomposable character and may alternatively be represented with a base letter followed by one or more general diacritical mark characters found in the Combining Diacritical Marks block.

The non-decomposable characters in this block, particularly in the range U+1EFA..U+1EFF, are mostly specialized letters used in Latin medieval manuscript traditions. These characters complement the larger set of medieval manuscript characters encoded in the Latin Extended-D block.

Capital Sharp S. U+1E9E LATIN CAPITAL LETTER SHARP S is for use in German. It is limited to specialized circumstances, such as uppercased strings in shop signage and book titles. The casing behavior of this character is unusual, as the recommended uppercase form for most casing operations on U+00DF ß LATIN SMALL LETTER SHARP S continues to be "SS". See the discussion of tailored casing in _Section 3.13, Default Case Algorithms_, for more about the casing of this character.

Vietnamese Vowel Plus Tone Mark Combinations. A portion of this block (U+1EA0.. U+1EF9) comprises vowel letters of the modern Vietnamese alphabet (*quốc ngũy*) combined with a diacritical mark that denotes the phonemic tone that applies to the syllable.

7.1.8 Latin Extended-C: U+2C60−U+2C7F

This small block of additional Latin characters contains orthographic Latin additions for minority languages, a few historic Latin letters, and further extensions for phonetic notations, particularly UPA.

Uyghur. The Latin orthography for the Uyghur language was influenced by widespread conventions for extension of the Cyrillic script for representing Central Asian languages. In particular, a number of Latin characters were extended with a Cyrillic-style descender diacritic to create new letters for use with Uyghur.

Claudian Letters. The Roman emperor Claudius invented three additional letters for use with the Latin script. Those letters saw limited usage during his reign, but were abandoned soon afterward. The *half h* letter is encoded in this block. The other two letters are encoded in other blocks: U+2132 TURNED CAPITAL F and U+2183 ROMAN NUMERAL REVERSED ONE HUNDRED (unified with the Claudian letter *reversed c*). Claudian letters in inscriptions are uppercase only, but may be transcribed by scholars in lowercase.

7.1.9 Latin Extended-D: U+A720−U+A7FF

This block contains a variety of historic letters for the Latin script and other uncommon phonetic and orthographic extensions to the script.

Egyptological Transliteration. The letters in the range U+A722..U+A725 are specialized letters used for the Latin transliteration of *alef* and *ain* in ancient Egyptian texts. Their forms are related to the modifier letter half rings (U+02BE..U+02BF) which are sometimes used in Latin transliteration of Arabic.

U+A7BD LATIN SMALL LETTER GLOTTAL I is another specialized letter, used for the Latin transliteration of *yod* in ancient Egyptian texts. It is also widely referred to as *Egyptological yod*. When used in an Egyptian transliteration context, this character is always displayed in italic form. An uppercase form may also occur in transliterations that follow Latin casing conventions. The *glottal i* also occurs in Latin transliteration of Ugaritic texts, along with the related *glottal a* and *glottal u* in this range, U+A7BA..U+A7BF. Prior to Version 12.0, users employed combinations of the base letters *i*, *a*, and *u* with combining diacritics, such as U+0313 COMBINING COMMA ABOVE, U+0357 COMBINING RIGHT HALF RING ABOVE, or U+0486 COMBINING CYRILLIC PSILI PNEUMATA. Use of the glottal letters in the range U+A7BA..U+A7BF is encouraged to provide consistent representation and better typographic results.

Historic Mayan Letters. The letters in the range U+A726..U+A72F are obsolete historic letters seen only in a few early Spanish manuscripts of Mayan languages. They are not used in modern Mayan orthographies.

European Medievalist Letters. The letters in the ranges U+A730..U+A778 and U+A7C2..U+A7C3 occur in a variety of European medievalist manuscript traditions. None of these have any modern orthographic usage. A number of these letterforms constitute abbreviations, often for common Latin particles or suffixes.

Insular and Celticist Letters. The Insular manuscript tradition was current in Anglo-Saxon England and Gaelic Ireland throughout the early Middle Ages. The letters *d*, *f*, *g*, *r*, *s*, and *t* had unique shapes in that tradition, different from the Carolingian letters used in the modern Latin script. Although these letters can be considered variant forms of ordinary Latin letters, they are separately encoded because of their use by antiquarian Edward Lhuyd in his 1707 work *Archæologia Britannica*, which described the Late Cornish language in a phonetic alphabet using these Insular characters. Other specialists may make use of these letters contrastively in Old English or Irish manuscript contexts or in secondary material discussing such manuscripts.

Orthographic Letter Additions. The letters and modifier letters in the range U+A788..U+A78C occur in modern orthographies of a few small languages of Africa, Mexico, and New Guinea. Several of these characters were based on punctuation characters originally, so their shapes are confusingly similar to ordinary ASCII punctuation. Because of this potential confusion, their use is not generally recommended outside the specific context of the few orthographies already incorporating them.

The modern orthographies of the Luiseño and Cupeño language communities in California use U+A7CC Ꟍ ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ s ᴡɪᴛʜ ᴅɪᴀɢᴏɴᴀʟ sᴛʀᴏᴋᴇ and U+A7CD ꟍ ʟᴀᴛɪɴ sᴍᴀʟʟ ʟᴇᴛᴛᴇʀ s ᴡɪᴛʜ ᴅɪᴀɢᴏɴᴀʟ sᴛʀᴏᴋᴇ. The shapes of these characters are distinct from those used for U+A7A8 Ꞩ ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ s ᴡɪᴛʜ ᴏʙʟɪǫᴜᴇ sᴛʀᴏᴋᴇ and U+A7A9 ꞩ ʟᴀᴛɪɴ sᴍᴀʟʟ ʟᴇᴛᴛᴇʀ s ᴡɪᴛʜ ᴏʙʟɪǫᴜᴇ sᴛʀᴏᴋᴇ, which are used instead for an old Lower Sorbian orthography.

Several Wakashan and Salishan languages of North America have bicameral orthographies. To accommodate casing practices in those orthographies, two Latin lambda characters are encoded in this block: U+A7DA Ꟛ ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ ʟᴀᴍʙᴅᴀ and U+A7DB ꟛ ʟᴀᴛɪɴ sᴍᴀʟʟ ʟᴇᴛᴛᴇʀ ʟᴀᴍʙᴅᴀ. Additionally, U+A7DC Ꟶ ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ ʟᴀᴍʙᴅᴀ ᴡɪᴛʜ sᴛʀᴏᴋᴇ is the uppercase form of U+019B ƛ ʟᴀᴛɪɴ sᴍᴀʟʟ ʟᴇᴛᴛᴇʀ ʟᴀᴍʙᴅᴀ ᴡɪᴛʜ sᴛʀᴏᴋᴇ. The glyphs for the uppercase Latin lambda and lambda with stroke characters are very distinct from that of the Greek uppercase lambda, as shown below. Glyphs for the lowercase Latin lambda characters U+A7DB and U+019B may also vary from the Greek shapes.

Greek lambda	U+039B Λ	U+03BB λ
Latin lambda	U+A7DA Ꟛ	U+A7DB ꟛ
Latin lambda with stroke	U+A7DC Ꟶ	U+019B ƛ

In these North American orthographies, as well as some other orthographies which are not bicameral, the Latin lambda with stroke characters, which represent a lateral affricate [tɬ], can also be glottalized. Glottalization is typically indicated with a raised comma. The recommended mark to represent this raised comma can vary, based on local orthographical practice and stylistic conventions, and may also be affected by the presence of other marks above the base letters, such as a hacek. Some examples are shown below.

	U+0313	U+0315	U+02BC	U+2019
alveolar stop	ṭ	ƫ	t'	t'
palatoalveolar affricate	č̓	č̓	č'	č'
lateral affricate	ƛ̓	ƛ̓	ƛ'	ƛ'

Sinological Dot. U+A78F ʟᴀᴛɪɴ ʟᴇᴛᴛᴇʀ sɪɴᴏʟᴏɢɪᴄᴀʟ ᴅᴏᴛ is a middle dot used in the sinological tradition to represent a glottal stop. This convention of representing a glottal stop with a middle dot was introduced by Bernhard Karlgren in the early 20th century for Middle Chinese reconstructions, and was adopted by other influential sinologists and Tangutologists. This dot is also used in Latin transliterations of Phags-pa text.

The representative glyph for U+A78F is larger than a typical middle dot used as punctuation, to avoid visual confusion with U+00B7 ᴍɪᴅᴅʟᴇ ᴅᴏᴛ. Use of the sino-

logical dot should be limited to the appropriate scholarly contexts; it is not intended as a letter substitution for other functions of U+00B7 MIDDLE DOT.

Early Pinyin Letters. Early, experimental drafts of Pinyin included a number of Latin letters with retroflex or palatal hooks, for example, U+0282 LATIN SMALL LETTER S WITH HOOK. These letters were not adopted in standard Pinyin, but are attested in the early documents and in discussions about the history of Pinyin. Because Pinyin allows for Latin capitalization conventions, those letters with hooks also occurred in uppercase forms. The uppercase letters in the range U+A7C4..U+A7C6 are encoded to represent those uppercase forms of early Pinyin letters with hooks.

Latvian Letters. The letters with strokes in the range U+A7A0..U+A7A9 are for use in the pre-1921 orthography of Latvian. During the 19th century and early 20th century, Latvian was usually typeset in a Fraktur typeface. Because Fraktur typefaces do not work well with detached diacritical marks, the extra letters required for Latvian were formed instead with overstruck bars. The new orthography introduced in 1921 replaced these letters with the current Latvian letters with cedilla diacritics. The *barred s* letters were also used in Fraktur representation of Lower Sorbian until about 1950.

Ancient Roman Epigraphic Letters. There are a small number of additional Latin epigraphic letters known from Ancient Roman inscriptions. These letters only occurred as monumental capitals in the inscriptions, and were not part of the regular Latin alphabet which later developed case distinctions.

7.1.10 Latin Extended-E: U+AB30–U+AB6F

This block contains a number of Latin letters and modifier letters for phonetic transcription systems. The majority of these are letters specifically associated with the Böhmer-Ascoli transcription system, more generally known as "Teuthonista." The Teuthonista system was extensively used in the 20th century to transcribe Germanic dialects. Teuthonista or closely related systems were also used in Switzerland and Italy to transcribe Romance dialects. For related characters, see the Combining Diacritical Marks Extended block, which contains a number of specialized combining diacritics for use in Teuthonista.

The Latin Extended-E block also contains a few rarely used letters from other transcription systems, including conventions used in Sino-Tibetan studies.

7.1.11 Latin Extended-F: U+10780–U+107BF

The Latin Extended-F block contains modifier letters used in phonetic transcription. Most of the characters in this block derive from current or retired letters of the International Phonetic Alphabet (IPA) that have been superscripted to indicate secondary

articulation, as well as lightly or incompletely articulated sounds. This block also includes a Voice Quality Symbol (VoQS) used in transcribing disordered speech.

7.1.12 Latin Extended-G: U+1DF00–U+1DFFF

This block contains a number of phonetic symbols from extended IPA (extIPA) used to transcribe disordered speech. The block also includes characters with hooks that appear in linguistic descriptions. In addition, a set of five click letters created by the phonetician Douglas Beach are included at U+1DF0B..U+1DF0F.

7.1.13 Latin Ligatures: U+FB00–U+FB06

This range in the Alphabetic Presentation Forms block (U+FB00..U+FB4F) contains several common Latin ligatures, which occur in legacy encodings. Whether to use a Latin ligature is a matter of typographical style as well as a result of the orthographical rules of the language. Some languages prohibit ligatures across word boundaries. In these cases, it is preferable for the implementations to use unligated characters in the backing store and provide out-of-band information to the display layer where ligatures may be placed.

Some format controls in the Unicode Standard can affect the formation of ligatures. See "Cursive Connection and Ligatures" in _Section 23.2, Layout Controls_.

7.2 Greek

7.2.1 Greek: U+0370–U+03FF

The Greek script is used for writing the Greek language. The Greek script had a strong influence on the development of the Latin, Cyrillic, and Coptic scripts.

The Greek script is written in linear sequence from left to right with the frequent use of nonspacing marks. There are two styles of such use: monotonic, which uses a single mark called *tonos*, and polytonic, which uses multiple marks. Greek letters come in uppercase and lowercase pairs. Spaces are used to separate words and provide the primary line breaking opportunities. Archaic Greek texts do not use spaces.

Standards. The Unicode encoding of Greek is based on ISO/IEC 8859-7, which is equivalent to the Greek national standard ELOT 928, designed for monotonic Greek. A number of variant and archaic characters are taken from the bibliographic standard ISO 5428.

Polytonic Greek. Polytonic Greek, used for ancient Greek (classical and Byzantine) and occasionally for modern Greek, may be encoded using either combining character sequences or precomposed base plus diacritic combinations. For the latter, see the following subsection, "Greek Extended: U+1F00–U+1FFF."

Nonspacing Marks. Several nonspacing marks commonly used with the Greek script are found in the Combining Diacritical Marks range (see *Table 7-3*).

Table 7-3. Nonspacing Marks Used with Greek

Code	Name	Alternative Names
U+0300	COMBINING GRAVE ACCENT	*varia*
U+0301	COMBINING ACUTE ACCENT	*tonos, oxia*
U+0304	COMBINING MACRON	
U+0306	COMBINING BREVE	
U+0308	COMBINING DIAERESIS	*dialytika*
U+0313	COMBINING COMMA ABOVE	*psili, smooth breathing mark*
U+0314	COMBINING REVERSED COMMA ABOVE	*dasia, rough breathing mark*
U+0342	COMBINING GREEK PERISPOMENI	*circumflex, tilde, inverted breve*
U+0343	COMBINING GREEK KORONIS	*comma above*
U+0345	COMBINING GREEK YPOGEGRAMMENI	*iota subscript*

Because the characters in the Combining Diacritical Marks block are encoded by shape, not by meaning, they are appropriate for use in Greek where applicable. The

character U+0344 COMBINING GREEK DIALYTIKA TONOS should not be used. The combination of *dialytika* plus *tonos* is instead represented by the sequence <U+0308 COMBINING DIAERESIS, U+0301 COMBINING ACUTE ACCENT>.

Multiple nonspacing marks applied to the same baseform character are encoded in inside-out sequence. See the general rules for applying nonspacing marks in *Section 2.11, Combining Characters*.

The basic Greek accent written in modern Greek is called *tonos*. It is represented by an acute accent (U+0301). The shape that the acute accent takes over Greek letters is generally steeper than that shown over Latin letters in Western European typographic traditions, and in earlier editions of this standard was mistakenly shown as a vertical line over the vowel. Polytonic Greek has several contrastive accents, and the accent, or *tonos*, written with an acute accent is referred to as *oxia*, in contrast to the *varia*, which is written with a grave accent.

U+0342 COMBINING GREEK PERISPOMENI may appear as a circumflex ̂, an inverted breve ̑, a tilde ̃, or occasionally a macron ̄. Because of this variation in form, the *perispomeni* was encoded distinctly from U+0303 COMBINING TILDE.

U+0313 COMBINING COMMA ABOVE and U+0343 COMBINING GREEK KORONIS both take the form of a raised comma over a baseform letter. U+0343 COMBINING GREEK KORONIS was included for compatibility reasons; U+0313 COMBINING COMMA ABOVE is the preferred form for general use. Greek uses guillemets for quotation marks; for Ancient Greek, the quotations tend to follow local publishing practice. Because of the possibility of confusion between smooth breathing marks and curly single quotation marks, the latter are best avoided where possible. When either breathing mark is followed by an acute or grave accent, the pair is rendered side-by-side rather than vertically stacked.

Accents are typically written above their base letter in an all-lowercase or all-uppercase word; they may also be omitted from an all-uppercase word. However, in a title-case word, accents applied to the first letter are commonly written to the left of that letter. This is a matter of presentation only—the internal representation is still the base letter followed by the combining marks. It is *not* the stand-alone version of the accents, which occur before the base letter in the text stream.

Iota. The nonspacing mark *ypogegrammeni* (also known as *iota subscript* in English) can be applied to the vowels *alpha*, *eta*, and *omega* to represent historic diphthongs. This mark appears as a small *iota* below the vowel. When applied to a single upper-case vowel, the iota does not appear as a subscript, but is instead normally rendered as a regular lowercase iota to the right of the uppercase vowel. This form of the iota is called *prosgegrammeni* (also known as *iota adscript* in English). In completely upper-cased words, the iota subscript should be replaced by a capital iota following the vowel. Precomposed characters that contain iota subscript or iota adscript also have

special mappings. (See *Section 5.18, Case Mappings*.) Archaic representations of Greek words, which did not have lowercase or accents, use the Greek capital letter iota following the vowel for these diphthongs. Such archaic representations require special case mapping, which may not be automatically derivable.

Variant Letterforms. U+03A5 GREEK CAPITAL LETTER UPSILON has two common forms: one looks essentially like the Latin capital Y, and the other has two symmetric upper branches that curl like rams' horns, "ϒ". The Y-form glyph has been chosen consistently for use in the code charts, both for monotonic and polytonic Greek. For mathematical usage, the rams' horn form of the glyph is required to distinguish it from the *Latin Y*. A third form is also encoded as U+03D2 GREEK UPSILON WITH HOOK SYMBOL (see *Figure 7-4*). The precomposed characters U+03D3 GREEK UPSILON WITH ACUTE AND HOOK SYMBOL and U+03D4 GREEK UPSILON WITH DIAERESIS AND HOOK SYMBOL should not normally be needed, except where necessary for backward compatibility for legacy character sets.

Figure 7-4. Variations in Greek Capital Letter Upsilon

$$\Upsilon \quad \Upsilon \quad \Upsilon$$

Variant forms of several other Greek letters are encoded as separate characters in this block. Often (but not always), they represent different forms taken on by the character when it appears in the final position of a word. Examples include U+03C2 GREEK SMALL LETTER FINAL SIGMA used in a final position and U+03D0 GREEK BETA SYMBOL, which is the form that U+03B2 GREEK SMALL LETTER BETA would take on in a medial or final position.

Of these variant letterforms, only *final sigma* should be used in encoding standard Greek text to indicate a final sigma. It is also encoded in ISO/IEC 8859-7 and ISO 5428 for this purpose. Because use of the final sigma is a matter of spelling convention, software should not automatically substitute a final form for a nominal form at the end of a word. However, when performing lowercasing, the final form needs to be generated based on the context. See *Section 3.13, Default Case Algorithms*.

In contrast, U+03D0 GREEK BETA SYMBOL, U+03D1 GREEK THETA SYMBOL, U+03D2 GREEK UPSILON WITH HOOK SYMBOL, U+03D5 GREEK PHI SYMBOL, U+03F0 GREEK KAPPA SYMBOL, U+03F1 GREEK RHO SYMBOL, U+03F4 GREEK CAPITAL THETA SYMBOL, U+03F5 GREEK LUNATE EPSILON SYMBOL, and U+03F6 GREEK REVERSED LUNATE EPSILON SYMBOL should be used only in mathematical formulas—never in Greek text. If positional or other shape differences are desired for these characters, they should be implemented by a font or rendering engine.

Representative Glyphs for Greek Phi. Starting with *The Unicode Standard, Version 3.0*, and the concurrent second edition of ISO/IEC 10646-1, the representative glyphs

for U+03C6 φ GREEK SMALL LETTER PHI and U+03D5 ϕ GREEK PHI SYMBOL were swapped compared to earlier versions. In ordinary Greek text, the character U+03C6 is used exclusively, although this character has considerable glyphic variation, sometimes represented with a glyph more like the representative glyph shown for U+03C6 φ (the "loopy" form) and less often with a glyph more like the representative glyph shown for U+03D5 ϕ (the "straight" form).

For mathematical and technical use, the straight form of the small phi is an important symbol and needs to be consistently distinguishable from the loopy form. The straight-form phi glyph is used as the representative glyph for the symbol phi at U+03D5 to satisfy this distinction.

The representative glyphs were reversed in versions of the Unicode Standard prior to Unicode 3.0. This resulted in the problem that the character explicitly identified as the mathematical symbol did not have the straight form of the character that is the preferred glyph for that use. Furthermore, it made it unnecessarily difficult for general-purpose fonts supporting ordinary Greek text to add support for Greek letters used as mathematical symbols. This resulted from the fact that many of those fonts already used the loopy-form glyph for U+03C6, as preferred for Greek body text; to support the phi symbol as well, they would have had to disrupt glyph choices already optimized for Greek text.

When mapping symbol sets or SGML entities to the Unicode Standard, it is important to make sure that codes or entities that require the straight form of the phi symbol be mapped to U+03D5 and not to U+03C6. Mapping to the latter should be reserved for codes or entities that represent the small phi as used in ordinary Greek text.

Fonts used primarily for Greek text may use either glyph form for U+03C6, but fonts that also intend to support technical use of the Greek letters should use the loopy form to ensure appropriate contrast with the straight form used for U+03D5.

Greek Letters as Symbols. The use of Greek letters for mathematical variables and operators is well established. Characters from the Greek block may be used for these symbols.

For compatibility purposes, a few Greek letters are separately encoded as symbols in other character blocks. Examples include U+00B5 µ MICRO SIGN in the Latin-1 Supplement character block and U+2126 Ω OHM SIGN in the Letterlike Symbols character block. The *ohm sign* is canonically equivalent to the *capital omega*, and normalization would remove any distinction. Its use is therefore discouraged in favor of *capital omega*. The same equivalence does not exist between *micro sign* and *mu*, and use of either character as a micro sign is common. For Greek text, only the *mu* should be used.

Symbols Versus Numbers. The characters *stigma*, *koppa*, and *sampi* are used only as numerals, whereas *archaic koppa* and *digamma* are used only as letters.

Compatibility Punctuation. Two specific modern Greek punctuation marks are encoded in the Greek and Coptic block: U+037E ";" GREEK QUESTION MARK and U+0387 "·" GREEK ANO TELEIA. The *Greek question mark* (or *erotimatiko*) has the shape of a semicolon, but functions as a question mark in the Greek script. The *ano teleia* has the shape of a middle dot, but functions as a semicolon in the Greek script.

These two compatibility punctuation characters have canonical equivalences to U+003B SEMICOLON and U+00B7 MIDDLE DOT, respectively; as a result, normalized Greek text will lose any distinctions between the Greek compatibility punctuation characters and the common punctuation marks. Furthermore, ISO/IEC 8859-7 and most vendor code pages for Greek simply make use of semicolon and middle dot for the punctuation in question. Therefore, use of U+037E and U+0387 is not necessary for interoperating with legacy Greek data, and their use is not generally encouraged for representation of Greek punctuation.

Historic Letters. Historic Greek letters have been retained from ISO 5428.

Coptic-Unique Letters. In the Unicode Standard prior to Version 4.1, the Coptic script was regarded primarily as a stylistic variant of the Greek alphabet. The letters unique to Coptic were encoded in a separate range at the end of the Greek character block. Those characters were to be used together with the basic Greek characters to represent the complete Coptic alphabet. Coptic text was supposed to be rendered with a font using the Coptic style of depicting the characters it shared with the Greek alphabet. Texts that mixed Greek and Coptic languages using that encoding model could be rendered only by associating an appropriate font by language.

The Unicode Technical Committee and ISO/IEC JTC1/SC2 determined that Coptic is better handled as a separate script. Starting with Unicode 4.1, a new Coptic block added all the letters formerly unified with Greek characters as separate Coptic characters. (See *Section 7.3, Coptic*.) Implementations that supported Coptic under the previous encoding model may, therefore, need to be modified. Coptic fonts may need to continue to support the display of both the Coptic and corresponding Greek character with the same shape to facilitate their use with older documents.

Related Characters. For math symbols, see *Section 22.5, Mathematical Symbols*. For additional punctuation to be used with this script, see C0 Controls and ASCII Punctuation (U+0000..U+007F).

7.2.2 Greek Extended: U+1F00–U+1FFF

The characters in this block constitute a number of precomposed combinations of Greek letters with one or more general diacritical marks; in addition, a number of

spacing forms of Greek diacritical marks are provided here. In particular, these characters can be used for the representation of polytonic Greek texts without the use of combining marks. Because they do not cover all possible combinations in use, some combining character sequences may be required for a given text.

Each of the letters contained in this block may be alternatively represented with a base letter from the Greek block followed by one or more general diacritical mark characters found in the Combining Diacritical Marks block.

Spacing Diacritics. Sixteen additional spacing diacritical marks are provided in this character block for use in the representation of polytonic Greek texts. Each has an alternative representation for use with systems that support nonspacing marks. The nonspacing alternatives appear in *Table 7-4*. The spacing forms are meant for keyboards and pedagogical use and are not to be used in the representation of title-case words. The compatibility decompositions of these spacing forms consist of the sequence U+0020 SPACE followed by the nonspacing form equivalents shown in *Table 7-4*.

<div align="center">

Table 7-4. Greek Spacing and Nonspacing Pairs

</div>

Spacing Form	Nonspacing Form
1FBD GREEK KORONIS	0313 COMBINING COMMA ABOVE
037A GREEK YPOGEGRAMMENI	0345 COMBINING GREEK YPOGEGRAMMENI
1FBF GREEK PSILI	0313 COMBINING COMMA ABOVE
1FC0 GREEK PERISPOMENI	0342 COMBINING GREEK PERISPOMENI
1FC1 GREEK DIALYTIKA AND PERISPOMENI	0308 COMBINING DIAERESIS + 0342 COMBINING GREEK PERISPOMENI
1FCD GREEK PSILI AND VARIA	0313 COMBINING COMMA ABOVE + 0300 COMBINING GRAVE ACCENT
1FCE GREEK PSILI AND OXIA	0313 COMBINING COMMA ABOVE + 0301 COMBINING ACUTE ACCENT
1FCF GREEK PSILI AND PERISPOMENI	0313 COMBINING COMMA ABOVE + 0342 COMBINING GREEK PERISPOMENI
1FDD GREEK DASIA AND VARIA	0314 COMBINING REVERSED COMMA ABOVE + 0300 COMBINING GRAVE ACCENT
1FDE GREEK DASIA AND OXIA	0314 COMBINING REVERSED COMMA ABOVE + 0301 COMBINING ACUTE ACCENT
1FDF GREEK DASIA AND PERISPOMENI	0314 COMBINING REVERSED COMMA ABOVE + 0342 COMBINING GREEK PERISPOMENI
1FED GREEK DIALYTIKA AND VARIA	0308 COMBINING DIAERESIS + 0300 COMBINING GRAVE ACCENT
1FEE GREEK DIALYTIKA AND OXIA	0308 COMBINING DIAERESIS + 0301 COMBINING ACUTE ACCENT
1FEF GREEK VARIA	0300 COMBINING GRAVE ACCENT
1FFD GREEK OXIA	0301 COMBINING ACUTE ACCENT

| 1FFE GREEK DASIA | 0314 COMBINING REVERSED COMMA ABOVE |

7.2.3 Ancient Greek Numbers: U+10140–U+1018F

Ancient Greeks primarily used letters of the Greek alphabet to represent numbers. However, some extensions to this usage required quite a few nonalphabetic symbols or symbols derived from letters. Those symbols are encoded in the Ancient Greek Numbers block.

Acrophonic Numerals. Greek acrophonic numerals are found primarily in ancient inscriptions from Attica and other Greek regions. *Acrophonic* means that the character used to represent each number is the initial letter of the word by which the number is called—for instance, H for "HECATON" = 100.

The Attic acrophonic system, named for the greater geographic area that includes the city of Athens, is the most common and well documented. The characters in the Ancient Greek Numbers block cover the Attic acrophonic numeral system as well as non-Attic characters that cannot be considered glyph variants of the Attic acrophonic repertoire. They are the standard symbols used to represent weight or cost, and they appear consistently in modern editions and scholarly studies of Greek inscriptions. Uppercase Greek letters from the Greek block are also used for acrophonic numerals.

The Greek acrophonic number system is similar to the Roman one in that it does not use decimal position, does not require a placeholder for zero, and has special symbols for 5, 50, 500, and so on. The system is language specific because of the acrophonic principle. In some cases the same symbol represents different values in different geographic regions. The symbols are also differentiated by the unit of measurement —for example, talents versus staters.

Other Numerical Symbols. Other numerical symbols encoded in the range U+10175..U+1018A appear in a large number of ancient papyri. The standard symbols used for the representation of numbers, fractions, weights, and measures, they have consistently been used in modern editions of Greek papyri as well as various publications related to the study and interpretation of ancient documents. Several of these characters have considerable glyphic variation. Some of these glyph variants are similar in appearance to other characters.

Symbol for Zero. U+1018A GREEK ZERO SIGN occurs whenever a sexagesimal notation is used in historical astronomical texts to record degrees, minutes and seconds, or hours, minutes and seconds. The most common form of zero in the papyri is a small circle with a horizontal stroke above it, but many variations exist. These are taken to be scribal variations and are considered glyph variants.

7.3 Coptic

7.3.1 Coptic: U+2C80–U+2CFF

The Coptic script is the final stage in the development of the Egyptian writing system. Coptic was subject to strong Greek influences because Greek was more identified with the Christian tradition, and the written demotic Egyptian no longer matched the spoken language. The Coptic script was based on the Greek uncial alphabets with several Coptic additional letters unique to Coptic. The Coptic language died out in the fourteenth century, but it is maintained as a liturgical language by Coptic Christians. Coptic is written from left to right in linear sequence; in modern use, spaces are used to separate words and provide the primary line breaking opportunities.

Prior to Version 4.1, the Unicode Standard treated Coptic as a stylistic variant of Greek. Seven letters unique to Coptic (14 characters with the case pairs) were encoded in the Greek and Coptic block. In addition to these 14 characters, Version 4.1 added a Coptic block containing the remaining characters needed for basic Coptic text processing. This block also includes standard logotypes used in Coptic text as well as characters for Old Coptic and Nubian.

Development of the Coptic Script. The best-known Coptic dialects are Sahidic and Bohairic. Coptic scholarship recognizes a number of other dialects that use additional characters. The repertoires of Sahidic and Bohairic reflect efforts to standardize the writing of Coptic, but attempts to write the Egyptian language with the Greek script preceded that standardization by several centuries. During the initial period of writing, a number of different solutions to the problem of representing non-Greek sounds were made, mostly by borrowing letters from Demotic writing. These early efforts are grouped by Copticists under the general heading of Old Coptic.

Casing. Coptic is considered a bicameral script. Historically, it was caseless, but it has acquired case through the typographic developments of the last centuries. Already in Old Coptic manuscripts, letters could be written larger, particularly at the beginning of paragraphs, although the capital letters tend to have the most distinctive shapes in the Bohairic tradition. To facilitate scholarly and other modern casing operations, Coptic has been encoded as a bicameral script, including uniquely Old Coptic characters.

Font Styles. Bohairic Coptic uses only a subset of the letters in the Coptic repertoire. It also uses a font style distinct from that for Sahidic. Prior to Version 5.0, the Coptic letters derived from Demotic, encoded in the range U+03E2..U+03EF in the Greek and Coptic block, were shown in the code charts in a Bohairic font style. Starting

from Version 5.0, all Coptic letters in the standard, including those in the range U+03E2..U+03EF, are shown in the code charts in a Sahidic font style, instead.

Characters for Cryptogrammic Use. U+2CB7 COPTIC SMALL LETTER CRYPTOGRAMMIC EIE and U+2CBD COPTIC SMALL LETTER CRYPTOGRAMMIC NI are characters for cryptogrammic use. A common Coptic substitution alphabet that was used to encrypt texts had the disadvantageous feature whereby three of the letters (*eie*, *ni*, and *fi*) were substituted by themselves. However, because *eie* and *ni* are two of the highest-frequency characters in Coptic, Copts felt that the encryption was not strong enough, so they replaced those letters with these cryptogrammic ones. Two additional cryptogrammic letters in less frequent use are also encoded: U+2CEC COPTIC SMALL LETTER CRYPTOGRAMMIC SHEI and U+2CEE COPTIC SMALL LETTER CRYPTOGRAMMIC GANGIA. Copticists preserve these letter substitutions in modern editions of these encrypted texts and do not consider them to be glyph variants of the original letters.

U+2CC0 COPTIC CAPITAL LETTER SAMPI has a numeric value of 900 and corresponds to U+03E0 GREEK LETTER SAMPI. It is not found in abecedaria, but is used in cryptogrammic contexts as a letter.

Crossed Shei. U+2CC3 ⳃ COPTIC SMALL LETTER CROSSED SHEI is found in Dialect I of Old Coptic, where it represents a sound /ç/. It is found alongside U+03E3 ϣ COPTIC SMALL LETTER SHEI, which represents /ʃ/. The diacritic is not productive.

Supralineation. In Coptic texts, a line is often drawn across the top of two or more characters in a row. There are two distinct conventions for this supralineation, each of which is represented by different sequences of combining marks.

The first of these is a convention for abbreviation, in which words are shortened by removal of certain letters. A line is then drawn across the tops of all of the remaining letters, extending from the beginning of the first to the end of the last letter of the abbreviated form. This convention is represented by following each character of the abbreviated form with U+0305 COMBINING OVERLINE. When rendered together, these combining overlines should connect into a continuous line.

The other convention is to distinguish the spelling of certain common words or to highlight proper names of divinities and heroes—a convention related to the use of cartouches in hieroglyphic Egyptian. In this case the supralineation extends from the *middle* of the first character in the sequence to the *middle* of the last character in the sequence. Instead of using U+0305 COMBINING OVERLINE for the entire sequence, one uses U+FE24 COMBINING MACRON LEFT HALF after the first character, U+FE25 COMBINING MACRON RIGHT HALF after the last character, and U+FE26 COMBINING CONJOINING MACRON after any intervening characters. This gives the effect of a line starting and ending in the middle of letters, rather than at their edges.

Combining Diacritical Marks. Bohairic text uses a mark called *jinkim* to represent syllabic consonants, which is indicated by either U+0307 COMBINING DOT ABOVE or U+0300 COMBINING GRAVE ACCENT. Other dialects, including Sahidic, use U+0304 COMBINING MACRON for the same purpose. A number of other generic diacritical marks are used with Coptic.

U+2CEF COPTIC COMBINING NI ABOVE is a script-specific combining mark, typically used at the end of a line to indicate a final *ni* after a vowel. In rendering, this mark typically hangs over the space to the right of its base character.

The characters U+2CF0 COPTIC COMBINING SPIRITUS ASPER and U+2CF1 COPTIC COMBINING SPIRITUS LENIS are analogues of the Greek breathing marks. They are used rarely in Coptic. When used, they typically occur over the letter U+2C8F COPTIC SMALL LETTER HATE, sometimes to indicate that it is the borrowed Greek conjunction "or", written with the cognate Greek letter *eta*.

Punctuation. Coptic texts use common punctuation, including *colon*, *full stop*, *semicolon* (functioning, as in Greek, as a question mark), and *middle dot*. Quotation marks are found in edited texts. In addition, Coptic-specific punctuation occurs: U+2CFE COPTIC FULL STOP and U+2CFF COPTIC MORPHOLOGICAL DIVIDER. Several other historic forms of punctuation are known only from Old Nubian texts.

Numerical Use of Letters. Numerals are indicated with letters of the alphabet, as in Greek. Sometimes the numerical use is indicated specifically by marking a line above, represented with U+0305 COMBINING OVERLINE. U+0375 GREEK LOWER NUMERAL SIGN or U+033F COMBINING DOUBLE OVERLINE can be used to indicate multiples of 1,000, as shown in *Figure 7-5*.

Figure 7-5. Coptic Numerals

Coptic	Value
ⲁ	1
ⲁ͵ or ⲁ̿	1,000
ⲁ͵ⲱⲡⲏ	1,888

U+0374 GREEK NUMERAL SIGN is used to indicate fractions. For example, ⲅʹ indicates the fractional value 1/3. There is, however, a special symbol for 1/2: U+2CFD COPTIC FRACTION ONE HALF.

7.4 Cyrillic

The Cyrillic script is one of several scripts that were ultimately derived from the Greek script. The details of the history of that development and of the relationship between early forms of writing systems for Slavic languages has been lost. Cyrillic has traditionally been used for writing various Slavic languages, among which Russian is predominant. The earliest attestations of Cyrillic are for Old Church Slavonic manuscripts, dating to the 10th century CE. Old Church Slavonic is also commonly referred to as Old Church Slavic, and is abbreviated as OCS.

In the nineteenth and early twentieth centuries, Cyrillic was extended to write the non-Slavic minority languages of Russia and neighboring countries.

Structure. The Cyrillic script is written in linear sequence from left to right with the occasional use of nonspacing marks. Cyrillic letters have uppercase and lowercase pairs. Spaces are used to separate words and provide the primary line breaking opportunities.

Historic Letterforms. The historic form of the Cyrillic alphabet—most notably that seen in Old Church Slavonic manuscripts—is treated as a font style variation of modern Cyrillic. The historic forms of the letters are relatively close to their modern appearance, and some of the historic letters are still in modern use in languages other than Russian. For example, U+0406 "I" CYRILLIC CAPITAL LETTER BYELORUS-SIAN-UKRAINIAN I is used in modern Ukrainian and Byelorussian, and is encoded amidst other modern Cyrillic extensions. Some of the historic letterforms were used in modern typefaces in Russian and Bulgarian. Prior to 1917, Russian made use of *yat*, *fita*, and *izhitsa*; prior to 1945, Bulgaria made use of these three as well as *big yus*.

Glagolitic. The particular early Slavic writing known as Glagolitic is treated as a distinct script from Cyrillic, rather than as a font style variation. The letterforms for Glagolitic, even though historically related, appear unrecognizably different from most modern Cyrillic letters. Glagolitic was also limited to a certain historic period; it did not grow to match the repertoire expansion of the Cyrillic script. See *Section 7.5, Glagolitic*.

7.4.1 Cyrillic: U+0400–U+04FF

Standards. The Cyrillic block of the Unicode Standard is based on ISO/IEC 8859-5.

Extended Cyrillic. These letters are used in alphabets for Turkic languages such as Azerbaijani, Bashkir, Kazakh, and Tatar; for Caucasian languages such as Abkhasian, Avar, and Chechen; and for Uralic languages such as Mari, Khanty, and Kildin Sami. The orthographies of some of these languages have often been revised in the past;

some of them have switched from Arabic to Latin to Cyrillic, and back again. Azerbaijani, for instance, is now officially using a Turkish-based Latin script.

Abkhasian. The Cyrillic orthography for Abkhasian has been updated fairly frequently over the course of the 20th and early 21st centuries. Some of these revisions involved changes in letterforms, often for the diacritic descenders used under extended Cyrillic letters for Abkhasian. The most recent such reform has been reflected in glyph changes for Abkhaz-specific Cyrillic letters in the code charts. In particular, U+04BF CYRILLIC SMALL LETTER ABKHASIAN CHE WITH DESCENDER, is now shown with a straight descender diacritic. In code charts for Version 5.1 and earlier, that character was displayed with a representative glyph using an ogonek-type hook descender, more typical of historic orthographies for Abkhasian. The glyph for U+04A9 CYRILLIC SMALL LETTER ABKHASIAN HA was also updated.

Other changes for Abkhasian orthography represent actual respellings of text. Of particular note, the character added in Version 5.2, U+0525 CYRILLIC SMALL LETTER PE WITH DESCENDER, is intended as a replacement for U+04A7 CYRILLIC SMALL LETTER PE WITH MIDDLE HOOK, which was used in older orthographies.

Palochka. U+04C0 "Ӏ" CYRILLIC LETTER PALOCHKA is used in Cyrillic orthographies for a number of Caucasian languages, such as Adyghe, Avar, Chechen, and Kabardian. The name *palochka* itself is based on the Russian word for "stick," referring to the shape of the letter. The glyph for *palochka* is usually indistinguishable from an uppercase Latin "I" or U+0406 "І" CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I; however, in some serifed fonts it may be displayed without serifs to make it more visually distinct.

In use, *palochka* typically modifies the reading of a preceding letter, indicating that it is an ejective. The *palochka* is generally caseless and should retain its form even in lowercased Cyrillic text. However, there is some evidence of distinctive lowercase forms; for those instances, U+04CF CYRILLIC SMALL LETTER PALOCHKA may be used.

Broad Omega. The name of U+047D CYRILLIC SMALL LETTER OMEGA WITH TITLO is anomalous. It does not actually have a *titlo*, but instead represents a *broad omega* with a *great apostrof* diacritic. (See U+A64D CYRILLIC SMALL LETTER BROAD OMEGA.) The *great apostrof* is a stylized diacritical mark consisting of the soft breathing mark (see U+0486 COMBINING CYRILLIC PSILI PNEUMATA) and the Cyrillic *kamora* (see U+0311 COMBINING INVERTED BREVE). Functionally, U+047D is analogous to the Greek character U+1F66 GREEK SMALL LETTER OMEGA WITH PSILI AND PERISPOMENI. Both the Greek and the Church Slavonic characters have identical functions—to record the exclamation "Oh!" U+047D is also known as the Cyrillic *beautiful omega*.

Digraph Onik and Monograph Uk. U+0479 CYRILLIC SMALL LETTER UK was intended for representation of the Church Slavonic *uk* vowel, which sometimes is rendered as a digraph *onik* form and sometimes as a monograph *uk* form. However, that ambiguity of rendering is not optimal for the representation of Church Slavonic text. The current recommendation is to avoid the use of U+0479, as well as its corresponding uppercase U+0478. The digraph *onik* has the preferred spelling consisting of the letter sequence <U+043E CYRILLIC SMALL LETTER O, U+0443 CYRILLIC SMALL LETTER U>. The monograph *uk* should be represented instead by an unambiguous letter intended specifically for that form: U+A64B CYRILLIC SMALL LETTER MONOGRAPH UK.

Palatalization. U+0484 COMBINING CYRILLIC PALATALIZATION is a diacritical mark used in ancient manuscripts and in academic work to indicate that a consonant is softened, a phenomenon called palatalization in Cyrillic studies. Although the shape of the diacritic is similar, this should not be confused with the use of U+0311 COMBINING INVERTED BREVE to represent the Cyrillic *kamora* (circumflex accent). Palatalization is also represented in some manuscripts and in academic publications with U+02BC MODIFIER LETTER APOSTROPHE or occasionally U+02B9 MODIFIER LETTER PRIME.

Combining Titlo. U+0483 COMBINING CYRILLIC TITLO is used in modern Church Slavonic to indicate that a letter or letters have been omitted from the spelling of a word (either in *nomina sacra* or in abbreviations). It also is used in numeral notation. In modern Church Slavonic it is not used to "cover" superscripted (*titlo*) letters; instead, U+0487 COMBINING CYRILLIC POKRYTIE is used as a cap over titlo letters. In Old Church Slavonic manuscripts, on the other hand, *pokrytie*, *titlo*, and its archaic typographical alternate U+A66F COMBINING CYRILLIC VZMET are all used more or less interchangeably.

7.4.2 Cyrillic Supplement: U+0500–U+052F

Komi. The characters in the range U+0500..U+050F are found in ISO 10754; they were used in Komi Cyrillic orthography from 1919 to about 1940. These letters use glyphs that differ structurally from other characters in the Unicode Standard that represent similar sounds—namely, Serbian ӎ and ӊ, which are ligatures of the base letters л and н with a palatalizing soft sign ь. The Molodtsov orthography made use of a different kind of palatalization hook for Komi ӎ, ӊ, ҭ, ԁ, and so on.

Kurdish Letters. Although the Kurdish language is almost always written in either the Arabic script or the Latin script, there also exists a Cyrillic orthography which saw some usage for Kurdish in the former Soviet Union. The Cyrillic letters *qa* and *we* in this block are encoded to enable the representation of Cyrillic Kurdish entirely in the Cyrillic script, without use of the similar Latin letters q and w, from which these Kurdish letters were ultimately derived.

7.4.3 Cyrillic Extended-A: U+2DE0–U+2DFF

Titlo Letters. This block contains a set of superscripted (written above), or *titlo*, letters, used in manuscript Old Church Slavonic texts and in modern Church Slavonic, usually to indicate abbreviations of words in the text. They can be found alone or in pairs that typically form digraphs or ligatures above one base character. These characters may be followed by U+0487 COMBINING CYRILLIC POKRYTIE in both old and modern texts. In Old Church Slavonic texts they may also be followed by U+0483 COMBINING CYRILLIC TITLO or its typographical alternate form, U+A66F COMBINING CYRILLIC VZMET. Modern Church Slavonic never uses the *titlo* mark to "cover" superscripted letters, and does not use the *vzmet* mark at all.

When used in combination, two titlo letters normally form a composite combining letter, in which the components appear side-by-side or ligated, a behavior which deviates from the default vertical stacking of multiple combining characters. Occasionally, titlo letters can also be found vertically stacked in Old Church Slavonic texts, in this case exhibiting default stacking behavior. As there is no semantic distinction associated with the two presentations, both are handled at the font level, without requiring the use of format characters. The usual ligated form and the less common vertical stacking of titlo letters are contrasted in *Figure 7-6* for the sequence <U+2DE3 COMBINING CYRILLIC LETTER DE, U+A675 COMBINING CYRILLIC LETTER I>.

Figure 7-6. Combination of Titlo Letters

A wide variety of composite titlo letters can be encountered in Old Church Slavonic manuscripts, including such combinations as *ghe-o*, *de-ie*, *de-i*, *de-o*, *de-uk*, *el-i*, *em-i*, *es-te*, and many others. One of these combinations has been encoded atomically in Unicode as U+2DF5 COMBINING CYRILLIC LETTER ES-TE. However, the preferred representation of a composite titlo *es-te* is the sequence <U+2DED COMBINING CYRILLIC LETTER ES, U+2DEE COMBINING CYRILLIC LETTER TE>.

The glyphs in the code chart for the Cyrillic Extended-A block are based on the modern Cyrillic letters to which these titlo letters correspond, but in Old Church Slavonic manuscripts, the actual glyphs used are related to the older forms of Cyrillic letters.

7.4.4 Cyrillic Extended-B: U+A640–U+A69F

This block contains an extended set of historic Cyrillic characters used in Old Cyrillic manuscript materials, particularly Old Church Slavonic.

Numeric Enclosing Signs. The combining numeric signs in the range U+A670..U+A672 extend the series of such combining signs from the main Cyrillic block. These enclosing signs were used around letters to indicate high decimal multiples of the basic numeric values of the letters.

Titlo Letters. Several additional titlo letters based on manuscript sources are encoded in the ranges U+A674..U+A67B and U+A69E..U+A69F. For a description of titlo letters, see the subsection "Cyrillic Extended-A: U+2DE0–U+2DFF" earlier in this section.

Old Abkhasian Letters. The letters in the range U+A680..U+A697 are obsolete letters for an old orthography of the Abkhaz language. These characters are no longer in use, and the Abkhaz language is currently represented using various Cyrillic extensions in the main Cyrillic block.

7.4.5 Cyrillic Extended-C: U+1C80–U+1C8F

This block contains a small collection of historic variants of common Cyrillic lowercase letters, as well as a some modern Cyrillic characters. The historic variants, located in the range U+1C80..U+1C88, are attested in early Church Slavonic printed books published between 1550 and 1700. Some of them also occur in books printed today by the Russian Old Ritualist communities and in books printed by the Russian Orthodox Church. No separate uppercase letters are encoded for these historic variants; they pair with the existing uppercase Cyrillic letters.

7.4.6 Cyrillic Extended-D: U+1E030–U+1E08F

The Cyrillic Extended-D block contains superscript and subscript characters that convey phonetic and phonological information. The superscript characters in this block are used for phonetic detail, in a manner analogous to IPA. They appear in academic and general-use dictionaries and linguistic descriptions, and can take diacritics, parallel to IPA usage. Three Cyrillic superscript modifiers are found in other blocks: U+A69C MODIFIER LETTER CYRILLIC HARD SIGN, U+A69D MODIFIER LETTER CYRILLIC SOFT SIGN, and U+1D78 MODIFIER LETTER CYRILLIC EN.

Some authors distinguish between superscript and subscript letters. In contrast to the superscript letters, Cyrillic subscript modifiers are used to denote phonological phenomena, specifically archigraphemes.

The block also includes a combining character, U+1E08F COMBINING CYRILLIC SMALL LETTER BYELORUSSIAN-UKRAINIAN I, which appears in modern language materials and in medieval texts.

7.5 Glagolitic

7.5.1 Glagolitic: U+2C00–U+2C5F

Glagolitic, from the Slavic root *glagol*, meaning "word," is an alphabet considered to have been devised by Saint Cyril in or around 862 CE for his translation of the Scriptures and liturgical books into Slavonic. The relatively few Glagolitic inscriptions and manuscripts that survive from this early period are of great philological importance. Glagolitic was eventually supplanted by the alphabet now known as Cyrillic.

Like Cyrillic, the Glagolitic script is written in linear sequence from left to right with no contextual modification of the letterforms. Spaces are used to separate words and provide the primary line breaking opportunities.

In parts of Croatia where a vernacular liturgy was used, Glagolitic continued in use until modern times: the last Glagolitic missal was printed in Rome in 1893 with a second edition in 1905. In these areas Glagolitic is still occasionally used as a decorative alphabet.

Glyph Forms. Glagolitic exists in two styles, known as round and square. Round Glagolitic is the original style and more geographically widespread, although surviving examples are less numerous. Square Glagolitic (and the cursive style derived from it) was used in Croatia from the thirteenth century. There are a few documents written in a style intermediate between the two. The letterforms used in the charts are round Glagolitic. Several of the letters have variant glyph forms, which are not encoded separately.

Ordering. The ordering of the Glagolitic alphabet is largely derived from that of the Greek alphabet, although nearly half the Glagolitic characters have no equivalent in Greek and not every Greek letter has its equivalent in Glagolitic.

Punctuation and Diacritics. Glagolitic texts use common punctuation, including *comma*, *full stop*, *semicolon* (functioning, as in Greek, as a question mark), and *middle dot*. In addition, several forms of multiple-dot, archaic punctuation occur, including U+2056 THREE DOT PUNCTUATION, U+2058 FOUR DOT PUNCTUATION, and U+2059 FIVE DOT PUNCTUATION. Quotation marks are found in edited texts. Glagolitic also used numerous diacritical marks, many of them shared in common with Cyrillic.

Numerical Use of Letters. Glagolitic letters have inherent numerical values. A letter may be rendered with a line above or a tilde above to indicate the numeric usage explicitly. Alternatively, U+00B7 MIDDLE DOT may be used, flanking a letter on both sides, to indicate numeric usage of the letter.

7.5.2 Glagolitic Supplement: U+1E000–U+1E02F

The Glagolitic Supplement block contains a set of Glagolitic *titlo* letters, used in the representation of letters written above other letters in Glagolitic manuscripts. The function and behavior of these letters is similar to that of *titlo* letters in the Cyrillic script. For further discussion, see "Titlo Letters" in *Section 7.4, Cyrillic*.

7.6 Armenian

7.6.1 Armenian: U+0530–U+058F

The Armenian script is used primarily for writing the Armenian language. It is written from left to right. Armenian letters have uppercase and lowercase pairs. Spaces are used to separate words and provide the primary line breaking opportunities.

The Armenian script was devised about 406 CE by Mesrop Maštoc' to give Armenians access to Christian scriptural and liturgical texts, which were otherwise available only in Greek and Syriac. The script has been used to write Classical or *Grabar* Armenian, Middle Armenian, and both of the literary dialects of Modern Armenian: East and West Armenian.

Orthography. Mesrop's original alphabet contained 30 consonants and 6 vowels in the following ranges:

U+0531..U+0554 *Ա...Ք Ayb* to *K'ē*

U+0561..U+0584 *ա...ք ayb* to *k'ē*

Armenian spelling was consistent during the *Grabar* period, from the fifth to the tenth centuries CE; pronunciation began to change in the eleventh century. In the twelfth century, the letters *ō* and *fē* were added to the alphabet to represent the diphthong [aw] (previously written *աւ aw*) and the foreign sound [f], respectively. The Soviet Armenian government implemented orthographic reform in 1922 and again in 1940, creating a difference between the traditional Mesropian orthography and what is known as Reformed orthography. The 1922 reform limited the use of *w* to the digraph *ow* (or *u*) and treated this digraph as a single letter of the alphabet.

User Community. The Mesropian orthography is presently used by West Armenian speakers who live in the diaspora and, rarely, by East Armenian speakers whose origins are in Armenia but who live in the diaspora. The Reformed orthography is used by East Armenian speakers living in the Republic of Armenia and, occasionally, by West Armenian speakers who live in countries formerly under the influence of the former Soviet Union. Spell-checkers and other linguistic tools need to take the differences between these orthographies into account, just as they do for British and American English.

Punctuation. Armenian makes use of a number of punctuation marks also used in other European scripts. Armenian words are delimited with spaces and may terminate on either a space or a punctuation mark. U+0589 ։ ARMENIAN FULL STOP, called *verǰakēt* in Armenian, is used to end sentences. A shorter stop functioning like the semicolon (like the *ano teleia* in Greek, but normally placed on the baseline like

U+002E FULL STOP) is called *miǰakēt*; it is represented by U+2024 . ONE DOT LEADER. U+055D ՝ ARMENIAN COMMA is actually used more as a kind of colon than as a comma; it combines the functionality of both elision and pause. Its Armenian name is *bowt'*. In Armenian dialect materials, U+0308 COMBINING DIAERESIS appears over the letters *ayb*, *oh*, and *vo*.

In Armenian it is possible to differentiate between word-joining and word-splitting hyphens. To join words, the *miowt'jan gic* - is used; it can be represented by either U+002D HYPHEN-MINUS or U+2010 ‐ HYPHEN. At the end of the line, to split words across lines, the *ent'amna* U+058A ֊ ARMENIAN HYPHEN may also be used. This character has a curved shape in some fonts, but a hyphen-like shape in others. Both the word-joiner and the word-splitter can also break at word boundaries, but the two characters have different semantics.

Several other punctuation marks are unique to Armenian, and these function differently from other kinds of marks. The tonal punctuation marks (U+055B ARMENIAN EMPHASIS MARK, U+055C ARMENIAN EXCLAMATION MARK, and U+055E ARMENIAN QUESTION MARK) are placed directly above and slightly to the right of the vowel whose sound is modified, instead of at the end of the sentence, as European punctuation marks are. Because of the mechanical limitations of some printing technologies, these punctuation marks have often been typographically rendered as spacing glyphs above and to the right of the modified vowel, but this practice is not recommended. Depending on the font, the kerning sometimes presents them as half-spacing glyphs, which is somewhat more acceptable. The placement of the Armenian tonal mark can be used to distinguish between different questions.

U+055F ARMENIAN ABBREVIATION MARK, or *patiw*, is one of four abbreviation marks found in manuscripts to abbreviate common words such as God, Jesus, Christos, Lord, Saint, and so on. It is placed above the abbreviated word and spans all of its letters.

Preferred Characters. The apostrophe at U+055A has the same shape and function as the Latin apostrophe at U+2019, which is preferred. There is no left half ring in Armenian. Unicode character U+0559 is not used. It appears that this character is a duplicate character, which was encoded to represent U+02BB MODIFIER LETTER TURNED COMMA, used in Armenian transliteration. U+02BB is preferred for this purpose.

Ligatures. Five Armenian ligatures are encoded in the Alphabetic Presentation Forms block in the range U+FB13..U+FB17. These shapes (along with others) are typically found in handwriting and in traditional fonts that mimic the manuscript ligatures. Of these, the *men-now* ligature is the one most useful for both traditional and modern fonts.

7.7 Georgian

7.7.1 Georgian: U+10A0–U+10FF

Georgian Extended: U+1C90–U+1CBF

Georgian Supplement: U+2D00–U+2D2F

The Georgian script is used primarily for writing the Georgian language and its dialects. It is also used for the Svan and Mingrelian languages and in the past was used for Abkhaz and other languages of the Caucasus. It is written from left to right. Spaces are used to separate words and provide the primary line breaking opportunities.

Script Forms. The script name "Georgian" in the Unicode Standard is used for what are really two closely related scripts. The original Georgian writing system was an inscriptional form called *Asomtavruli*, from which a manuscript form called *Nuskhuri* was derived. Together these forms are categorized as *Khutsuri* (ecclesiastical), in which *Asomtavruli* is used as the uppercase and *Nuskhuri* as the lowercase. This development of a bicameral script parallels the evolution of the Latin alphabet, in which the original linear monumental style became the uppercase and manuscript styles of the same alphabet became the lowercase. The *Khutsuri* script is still used for liturgical purposes, but was replaced, through a history now uncertain, by an alphabet called *Mkhedruli* (military), which is the form used for nearly all modern Georgian writing. The Georgian Mkhedruli alphabet has been fundamentally caseless since its development.

The scholar Akaki Shanidze attempted to introduce a casing practice for Georgian in the 1950s, but this system failed to gain popularity. In his typographic departure, he used the *Asomtavruli* forms to represent uppercase letters, alongside "lowercase" *Mkhedruli.*

Following this failed casing practice with *Asomtavruli* forms, *Mtavruli* forms developed as a particular style of *Mkhedruli* in which the distinction between letters with ascenders and descenders was not maintained. All letters written in the *Mtavruli* style appear with an equal height standing on the baseline, similar to small caps in the Latin script.

Version 11.0 of the Unicode standard added a set of *Mtavruli* letters at U+1C90..U+1CBF. These *Mtavruli* letters have a casing relationship defined with *Mkhedruli* letters: the *Mtavruli* letters are the uppercase forms of the *Mkhedruli* letters, which now are considered lowercase forms.

Figure 7-7 uses Akaki Shanidze's name to illustrate the various forms of Georgian text.

Figure 7-7. Georgian Scripts and Casing

Asomtavruli majuscule	ႤႡႠႩႨ ႸႠႬႨႻႤ
Nuskhuri minuscule	ⴄⴁⴀⴉⴈ ⴘⴀⴌⴈⴛⴄ
Casing Khutsuri	Ⴄⴁⴀⴉⴈ Ⴘⴀⴌⴈⴛⴄ
Mkhedruli	აკაკი შანიძე
Mtavruli	ᲐᲙᲐᲙᲘ ᲨᲐᲜᲘᲫᲔ

Both the modern *Mkhedruli* lowercase form and the *Asomtavruli* inscriptional form are encoded in the Georgian block. The *Nuskhuri* script form is encoded in the Georgian Supplement block, and the modern *Mtavruli* uppercase form is encoded in the Georgian Extended block.

Case Forms. For most of modern *Mkhedruli* writing, *Mtavruli* has been used as an emphatic or headline style. In Version 11.0 of the Unicode Standard, that usage was broadened to define formal case pair mappings between these forms, with *Mkhedruli* serving as lowercase and *Mtavruli* serving as uppercase. Georgian casing established in Version 11.0 does not extend to title casing, as the Georgian script does not have title casing for individual words or sentences. *Mtavruli* continues to be used as an emphatic and headline style.

The Unicode Standard also provides case mappings between the two Khutsuri forms: *Asomtavruli* and *Nuskhuri*.

Punctuation. Modern Georgian text uses generic European conventions for punctuation. See the common punctuation marks in the Basic Latin and General Punctuation blocks.

Historic Punctuation. Historic Georgian manuscripts, particularly text in the older, ecclesiastical styles, use manuscript punctuation marks common to the Byzantine tradition. These include single, double, and multiple dot punctuation. For a single dot punctuation mark, U+00B7 MIDDLE DOT or U+2E31 WORD SEPARATOR MIDDLE DOT may be used. Historic double and multiple dot punctuation marks can be found in the U+2056..U+205E range in the General Punctuation block and in the U+2E2A..U+2E2D range in the Supplemental Punctuation block.

U+10FB GEORGIAN PARAGRAPH SEPARATOR is a historic punctuation mark commonly used in Georgian manuscripts to delimit text elements comparable to a paragraph level. Although this punctuation mark may demarcate a paragraph in exposition, it does not force an actual paragraph termination in the text flow. To

cause a paragraph termination, U+10FB must be followed by a newline character, as described in *Section 5.8, Newline Guidelines*.

Prior to Version 6.0 the Unicode Standard recommended the use of U+0589 ARMENIAN FULL STOP as the two dot version of the full stop for historic Georgian documents. This is no longer recommended because designs for Armenian fonts may be inconsistent with the display of Georgian text, and because other, generic two dot punctuation characters are available in the standard, such as U+205A TWO DOT PUNCTUATION or U+003A COLON.

For additional punctuation to be used with this script, see C0 Controls and ASCII Punctuation (U+0000..U+007F) and General Punctuation (U+2000..U+206F).

7.8 Modifier Letters

Modifier letters, in the sense used in the Unicode Standard, are letters or symbols that are typically written adjacent to other letters and which modify their usage in some way. They are not formally combining marks (gc = Mn or gc = Mc) and do not *graphically* combine with the base letter that they modify. They are base characters in their own right. The sense in which they modify other letters is more a matter of their semantics in usage; they often tend to function as if they were diacritics, indicating a change in pronunciation of a letter, or otherwise distinguishing a letter's use. Typically this diacritic modification applies to the character preceding the modifier letter, but modifier letters may sometimes modify a following character. Occasionally a modifier letter may simply stand alone representing its own sound.

Modifier letters are commonly used in technical phonetic transcriptional systems, where they augment the use of combining marks to make phonetic distinctions. Some of them have been adapted into regular language orthographies as well. For example, U+02BB MODIFIER LETTER TURNED COMMA is used to represent the *'okina* (glottal stop) in the orthography for Hawaiian.

Many modifier letters take the form of superscript or subscript letters. Thus the IPA modifier letter that indicates labialization (U+02B7) is a superscript form of the letter *w*. As for all such superscript or subscript form characters in the Unicode Standard, these modifier letters have compatibility decompositions.

Case and Modifier Letters. Most modifier letters are derived from letters in the Latin script, although some modifier letters occur in other scripts. Latin-derived modifier letters may be based on either minuscule (lowercase) or majuscule (uppercase) forms of the letters, but never have case mappings. Modifier letters which have the shape of capital or small capital Latin letters, in particular, are used exclusively in technical phonetic transcriptional systems. Strings of phonetic transcription are notionally lowercase—all letters in them are considered to be lowercase, whatever their shapes. In terms of formal properties in the Unicode Standard, modifier letters based on letter shapes are Lowercase = True; modifier letters not based on letter shapes are simply caseless. All modifier letters, regardless of their shapes, are operationally caseless; they need to be unaffected by casing operations, because changing them by a casing operation would destroy their meaning for the phonetic transcription. Only those superscript or subscript forms that have specific usage in IPA, the Uralic Phonetic Alphabet (UPA), or other major phonetic transcription systems are encoded.

General Category. Modifier letters in the Unicode Standard are indicated by either one of two General_Category values: gc = Lm or gc = Sk. The General_Category Lm is given to modifier letters derived from regular letters. It is also given to some other characters with more punctuation-like shapes, such as raised commas, which never-

theless have letterlike behavior and which occur on occasion as part of the orthography for regular words in one language or another. The General_Category Sk is given to modifier letters that typically have more symbol-like origins and which seldom, if ever, are adapted to regular orthographies outside the context of technical phonetic transcriptional systems. This subset of modifier letters is also known as "modifier symbols."

This distinction between gc = Lm and gc = Sk is reflected in other Unicode specifications relevant to identifiers and word boundary determination. Modifier letters with gc = Lm are included in the set definitions that result in the derived properties ID_Start and ID_Continue (and XID_Start and XID_Continue). As such, they are considered part of the default definition of Unicode identifiers. Modifier *symbols* (gc = Sk), on the other hand, are *not* included in those set definitions, and so are excluded by default from Unicode identifiers.

Modifier letters (gc = Lm) have the derived property Alphabetic, while modifier symbols (gc = Sk) do not. Modifier letters (gc = Lm) also have the word break property value (wb = ALetter), while modifier symbols (gc = Sk) do not. This means that for default determination of word break boundaries, modifier symbols will cause a word break, while modifier letters proper will not.

Blocks. Most general use modifier letters (and modifier symbols) were collected together in the Spacing Modifier Letters block (U+02B0..U+02FF), the UPA-related Phonetic Extensions block (U+1D00..U+1D7F), the Phonetic Extensions Supplement block (U+1D80..U+1DBF), and the Modifier Tone Letters block (U+A700..U+A71F). However, some script-specific modifier letters are encoded in the blocks appropriate to those scripts. They can be identified by checking for their General_Category values.

Character Names. There is no requirement that the Unicode character names for modifier letters contain the label "MODIFIER LETTER", although most of them do.

7.8.1 Spacing Modifier Letters: U+02B0–U+02FF

Phonetic Usage. The majority of the modifier letters in this block are phonetic modifiers, including the characters required for coverage of the International Phonetic Alphabet. In many cases, modifier letters are used to indicate that the pronunciation of an adjacent letter is different in some way—hence the name "modifier." They are also used to mark stress or tone, or may simply represent their own sound. Many of these modifiers letters correspond to separate, nonspacing diacritical marks; the specific cross references can be found in the code charts.

Encoding Principles. This block includes characters that may have different semantic values attributed to them in different contexts. It also includes multiple characters that may represent the same semantic values—there is no necessary one-

to-one relationship. The intention of the Unicode encoding is not to resolve the variations in usage, but merely to supply implementers with a set of useful forms from which to choose. The list of usages given for each modifier letter should not be considered exhaustive. For example, the glottal stop (Arabic *hamza*) in Latin transliteration has been variously represented by the characters U+02BC MODIFIER LETTER APOSTROPHE, U+02BE MODIFIER LETTER RIGHT HALF RING, and U+02C0 MODIFIER LETTER GLOTTAL STOP. Conversely, an apostrophe can have several uses; for a list, see the entry for U+02BC MODIFIER LETTER APOSTROPHE in the character names list. There are also instances where an IPA modifier letter is explicitly equated in semantic value to an IPA nonspacing diacritic form.

Superscript Letters. Some of the modifier letters are superscript forms of other letters. The most commonly occurring of these superscript letters are encoded in this block, but many others, particularly for use in UPA, can be found in the Phonetic Extensions block (U+1D00..U+1D7F) and in the Phonetic Extensions Supplement block (U+1D80..U+1DBF). The superscript forms of the *i* and *n* letters can be found in the Superscripts and Subscripts block (U+2070..U+209F). The fact that the latter two letters contain the word "superscript" in their names instead of "modifier letter" is an historical artifact of original sources for the characters, and is not intended to convey a functional distinction in the use of these characters in the Unicode Standard.

Superscript modifier letters are intended for cases where the letters carry a specific meaning, as in phonetic transcription systems, and are not a substitute for generic styling mechanisms for superscripting of text, as for footnotes, mathematical and chemical expressions, and the like.

The superscript modifier letters are *spacing* letters, and should be distinguished from superscripted *combining* Latin letters. The superscripted combining Latin letters, as for example those encoded in the Combining Diacritical Marks block in the range U+0363..U+036F, are associated with the Latin historic manuscript tradition, often representing various abbreviatory conventions in text.

Spacing Clones of Diacritics. Some corporate standards explicitly specify spacing and nonspacing forms of combining diacritical marks, and the Unicode Standard provides matching codes for these interpretations when practical. A number of the spacing forms are included in the Basic Latin and Latin-1 Supplement blocks. The six common European diacritics that do not have spacing forms encoded in those blocks are encoded as spacing characters in the Spacing Modifier Letters block instead. These forms can have multiple semantics, such as U+02D9 DOT ABOVE, which is used as an indicator of the Mandarin Chinese fifth (neutral) tone.

Rhotic Hook. U+02DE MODIFIER LETTER RHOTIC HOOK is defined in IPA as a freestanding modifier letter. In common usage, it is treated as a ligated hook on a baseform letter. Hence U+0259 LATIN SMALL LETTER SCHWA + U+02DE MODIFIER

LETTER RHOTIC HOOK may be treated as equivalent to U+025A LATIN SMALL LETTER SCHWA WITH HOOK.

Tone Letters. U+02E5..U+02E9 comprises a set of basic tone letters defined in IPA and commonly used in detailed tone transcriptions of African and other languages. Each tone letter refers to one of five distinguishable tone levels. To represent contour tones, the tone letters are used in combinations. The rendering of contour tones follows a regular set of ligation rules that results in a graphic image of the contour (see *Figure 7-8*).

<div align="center">

Figure 7-8. Tone Letters

</div>

1 + 5	→	⍝ (rising contour)
5 + 1	→	⍖ (falling contour)
3 + 5	→	⍝ (high rising contour)
1 + 3	→	⍝ (low rising contour)
1 + 3 + 1	→	⍝ (rising-falling contour)

For example, the sequence "1 + 5" in the first row of *Figure 7-8* indicates the sequence of the lowest tone letter, U+02E9 MODIFIER LETTER EXTRA-LOW TONE BAR, followed by the highest tone letter, U+02E5 MODIFIER LETTER EXTRA-HIGH TONE BAR. In that sequence, the tone letter is drawn with a ligation from the iconic position of the low tone to that of the high tone to indicate the sharp rising contour. A sequence of three tone letters may also be ligated, as shown in the last row of *Figure 7-8*, to indicate a low rising-falling contour tone.

7.8.2 Modifier Tone Letters: U+A700–U+A71F

The Modifier Tone Letters block contains modifier letters used in various schemes for marking tones. These supplement the more commonly used tone marks and tone letters found in the Spacing Modifier Letters block (U+02B0..U+02FF).

The characters in the range U+A700..U+A707 are corner tone marks used in the transcription of Chinese. They were invented by Bridgman and Wells Williams in the 1830s. They have little current use, but are seen in a number of old Chinese sources.

The tone letters in the range U+A708..U+A716 complement the basic set of IPA tone letters (U+02E5..U+02E9) and are used in the representation of Chinese tones for the most part. The dotted tone letters are used to represent short ("stopped") tones. The left-stem tone letters are mirror images of the IPA tone letters; like those tone letters, they can be ligated in sequences of two or three tone letters to represent contour tones. Left-stem versus right-stem tone letters are sometimes used contrastively to distinguish between tonemic and tonetic transcription or to show the effects of tonal sandhi.

The modifier letters in the range U+A717..U+A71A indicate tones in a particular orthography for Chinantec, an Oto-Manguean language of Mexico. These tone marks are also spacing modifier letters and are not meant to be placed over other letters.

7.9 Combining Marks

Combining marks are a special class of characters in the Unicode Standard that are intended to combine with a preceding character, called their *base*. They have a formal syntactic relationship—or *dependence*—on their base, as defined by the standard. This relationship is relevant to the definition of combining character sequences, canonical reordering, and the Unicode Normalization Algorithm. For formal definitions, see *Section 3.6, Combination*.

Combining marks usually have a visible glyphic form, but some of them are invisible. When visible, a combining mark may interact graphically with neighboring characters in various ways. Visible combining marks are divided roughly into two types: nonspacing marks and spacing marks. In rendering, the nonspacing marks generally have no baseline advance of their own, but instead are said to *apply* to their *grapheme base*. Spacing marks behave more like separate letters, but in some scripts they may have complex graphical interactions with other characters. For an extended discussion of the principles for the application of combining marks, see *Section 3.6, Combination*.

Nonspacing marks come in two types: diacritic and other. The diacritics are exemplified by such familiar marks as the *acute accent* or the *macron*, which are applied to letters of the Latin script (or similar scripts). They tend to indicate a change in pronunciation or a particular tone or stress. They may also be used to derive new letters. However, in some scripts, such as Arabic and Hebrew, other kinds of nonspacing marks, such as *vowel points*, represent separate sounds in their own right and are not considered diacritics.

Sequence of Base Letters and Combining Marks. In the Unicode character encoding, all combining marks are encoded *after* their base character. For example, the Unicode character sequence U+0061 "a" LATIN SMALL LETTER A, U+0308 "◌̈" COMBINING DIAERESIS, U+0075 "u" LATIN SMALL LETTER U unambiguously encodes "äu", *not* "aü", as shown in *Figure 2-18*.

The Unicode Standard convention is consistent with the logical order of other nonspacing marks in Semitic and Indic scripts, the great majority of which follow the base characters with respect to which they are positioned. This convention is also in line with the way modern font technology handles the rendering of nonspacing glyphic forms, so that mapping from character memory representation to rendered glyphs is simplified. (For more information on the formal behavior of combining marks, see *Section 2.11, Combining Characters*, and *Section 3.6, Combination*.)

Multiple Semantics. Because nonspacing combining marks have such a wide variety of applications, they may have multiple semantic values. For example,

U+0308 = *diaeresis* = *trema* = *umlaut* = *double derivative*. Such multiple functions for a single combining mark are not separately encoded in the standard.

Glyphic Variation. When rendered in the context of a language or script, like ordinary letters, combining marks may be subjected to systematic stylistic variation, as discussed in *Section 7.1, Latin*. For example, when used in Polish, U+0301 COMBINING ACUTE ACCENT appears at a steeper angle than when it is used in French. When it is used for Greek (as *oxia*), it can appear nearly upright. U+030C COMBINING CARON is commonly rendered as an apostrophe when used with certain letterforms. U+0326 COMBINING COMMA BELOW is sometimes rendered as a *turned comma above* on a lowercase "g" to avoid conflict with the descender. In many fonts, there is no clear distinction made between U+0326 COMBINING COMMA BELOW and U+0327 COMBINING CEDILLA.

Combining accents above the base glyph are usually adjusted in height for use with uppercase versus lowercase forms. In the absence of specific font protocols, combining marks are often designed as if they were applied to typical base characters in the same font. However, this will result in suboptimal appearance in rendering and may cause security problems. See Unicode Technical Report #36, "Unicode Security Considerations."

For more information, see *Section 5.13, Rendering Nonspacing Marks*.

Overlaid Diacritics. A few combining marks are encoded to represent overlaid diacritics such as U+0335 COMBINING SHORT STROKE OVERLAY (= "bar") or hooks modifying the shape of base characters, such as U+0322 COMBINING RETROFLEX HOOK BELOW. Such overlaid diacritics are not used in decompositions of characters in the Unicode Standard. Overlaid combining marks for the indication of negation of mathematical symbols are an exception to this rule and are discussed later in this section.

One should use the combining marks for overlaid diacritics sparingly and with care, as rendering them on letters may create opportunities for spoofing and other confusion. Sequences of a letter followed by an overlaid diacritic or hook character are *not* canonically equivalent to any preformed encoded character with diacritic even though they may appear the same. See "Non-decomposition of Certain Diacritics" in *Section 2.12, Equivalent Sequences* for more discussion of the implications of overlaid diacritics for normalization and for text matching operations.

Marks as Spacing Characters. By convention, combining marks may be exhibited in (apparent) isolation by applying them to U+00A0 NO-BREAK SPACE. This approach might be taken, for example, when referring to the diacritical mark itself as a mark, rather than using it in its normal way in text. Prior to Version 4.1 of the Unicode Standard, the standard also recommended the use of U+0020 SPACE for display of isolated combining marks. This is no longer recommended, however,

because of potential conflicts with the handling of sequences of U+0020 SPACE characters in such contexts as XML.

In charts and illustrations in this standard, the combining nature of these marks is illustrated by applying them to a dotted circle, as shown in the examples throughout this standard.

In a bidirectional context, using any character with neutral directionality (that is, with a Bidirectional Class of ON, CS, and so on) as a base character, including U+00A0 NO-BREAK SPACE, a dotted circle, or any other symbol, can lead to unintended separation of the base character from certain types of combining marks during bidirectional ordering. The result is that the combining mark will be graphically applied to something other than the correct base. This affects spacing combining marks (that is, with a General Category of Mc) but not nonspacing combining marks. The unintended separation can be prevented by bracketing the combining character sequence with RLM or LRM characters as appropriate. For more details on bidirectional reordering, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Spacing Clones of Diacritical Marks. The Unicode Standard separately encodes clones of many common European diacritical marks, primarily for compatibility with existing character set standards. These cloned accents and diacritics are *spacing* characters and can be used to display the mark in isolation, without application to a NO-BREAK SPACE. They are cross-referenced to the corresponding combining mark in the names list in the Unicode code charts. For example, U+02D8 BREVE is cross-referenced to U+0306 COMBINING BREVE. Most of these spacing clones also have compatibility decomposition mappings involving U+0020 SPACE, but implementers should be cautious in making use of those decomposition mappings because of the complications that can arise from replacing a spacing character with a SPACE + combining mark sequence.

Relationship to ISO/IEC 8859-1. ISO/IEC 8859-1 contains eight characters that are ambiguous regarding whether they denote combining characters or separate spacing characters. In the Unicode Standard, the corresponding code points (U+005E ^ CIRCUMFLEX ACCENT, U+005F _ LOW LINE, U+0060 ` GRAVE ACCENT, U+007E ~ TILDE, U+00A8 ¨ DIAERESIS, U+00AF ‾ MACRON, U+00B4 ´ ACUTE ACCENT, and U+00B8 ¸ CEDILLA) are used only as spacing characters. The Unicode Standard provides unambiguous combining characters in the Combining Diacritical Marks block, which can be used to represent accented Latin letters by means of composed character sequences.

U+00B0 ° DEGREE SIGN is also occasionally used ambiguously by implementations of ISO/IEC 8859-1 to denote a spacing form of a diacritic ring above a letter; in the Unicode Standard, that spacing diacritical mark is denoted unambiguously by U+02DA ˚ RING ABOVE. U+007E "~" TILDE is ambiguous between usage as a spacing

form of a diacritic and as an operator or other punctuation; it is generally rendered with a center line glyph, rather than as a diacritic raised tilde. The spacing form of the diacritic tilde is denoted unambiguously by U+02DC "˜" SMALL TILDE.

Diacritics Positioned Over Two Base Characters. IPA, pronunciation systems, some transliteration systems, and a few languages such as Tagalog use diacritics that are applied to a sequence of two letters. This display of diacritics over two letters, also known as the use of *double diacritics*, is most often noted for the Latin script, which is widely used for transcription and transliteration. However, the use of double diacritics is not limited to the Latin script.

In rendering, these marks of unusual size appear as wide diacritics spanning across the top (or bottom) of the two base characters. The Unicode Standard contains a set of double-diacritic combining marks to represent such forms. Like all other combining nonspacing marks, these marks apply to the previous base character, but they are intended to hang over the following letter as well. For example, the character U+0360 COMBINING DOUBLE TILDE is intended to be displayed as depicted in *Figure 7-9*.

Figure 7-9. Double Diacritics

The Unicode Standard also contains a set of combining half diacritical marks, which can be used as an alternative, but not generally recommended, way of representing diacritics over a sequence of two (or more) letters. See "Combining Half Marks" later in this section and *Figure 7-15*.

The double-diacritical marks have a very high combining class—higher than all other nonspacing marks except U+0345 *iota subscript*—and so always are at or near the end of a combining character sequence when canonically reordered. In rendering, the double diacritic will float above other diacritics above (or below other diacritics below)—excluding surrounding diacritics—as shown in *Figure 7-10*.

Figure 7-10. Positioning of Double Diacritics

In *Figure 7-10*, the first line shows a combining character sequence in canonical order, with the double-diacritic tilde following a circumflex accent. The second line shows an alternative order of the two combining marks that is canonically equivalent to the first line. Because of this canonical equivalence, the two sequences should display identically, with the double diacritic floating above the other diacritics applied to single base characters.

Occasionally one runs across orthographic conventions that use a dot, an acute accent, or other simple diacritic *above* a *ligature tie*—that is, U+0361 COMBINING DOUBLE INVERTED BREVE. Because of the considerations of canonical order just discussed, one cannot represent such text simply by putting a *combining dot above* or *combining acute* directly after U+0361 in the text. Instead, the recommended way of representing such text is to place U+034F COMBINING GRAPHEME JOINER (CGJ) between the *ligature tie* and the combining mark that follows it, as shown in *Figure 7-11*.

Figure 7-11. Use of CGJ with Double Diacritics

$$u \; + \; \widehat{\bigcirc} \; + \; \boxed{\text{CGJ}} \; + \; \acute{\bigcirc} \; + \; i \; \rightarrow \; \widehat{\acute{ui}}$$

0075 0361 034F 0301 0069

Because CGJ has a combining class of zero, it blocks reordering of the double diacritic to follow the second combining mark in canonical order. The sequence of <CGJ, acute> is then rendered with default stacking, placing it centered above the *ligature tie*. This convention can be used to create similar effects with combining marks above other double diacritics (or below double diacritics that render below base characters).

For more information on the combining grapheme joiner, see "Combining Grapheme Joiner" in *Section 23.2, Layout Controls*.

Diacritics Positioned Over Three or More Base Characters. Some transcriptional systems extend the convention of double-diacritic display and show diacritics above (or below) three or more base letters. There are no characters encoded in the Unicode Standard which are specifically designated for plain text representation of triple diacritics. Instead, the recommendation of the Unicode Standard is to use text markup for such representation. The application of modifying text marks to arbitrary spans of text exceeds the normal scope of plain text and is usually better dealt with by conventions designed for rich text. In some limited circumstances, the combining half mark diacritics can be used in combinations to represent triple diacritics, but the

display of half mark diacritics used in this way often is unsatisfactory in plain text rendering.

Subtending Marks. An additional class of marks called *subtending marks* is positioned under (or occasionally over or surrounding) a sequence of several other characters. Formally, these marks are not treated as combining marks (gc = M), but instead as format characters (gc = Cf). In the text representation, they *precede* the sequence of characters they subtend, rather than follow a single base character, as combining marks do.

Although the terms s*ubtending marks* and *prefixed format control characters* have been used for these special marks for a number of versions of the Unicode Standard, as of Version 9.0 another more precise but equivalent term has been introduced for them: *prepended concatenation marks.* That term focuses on the order of occurrence of the marks (prepended to the sequence following them in the backing store), rather than the graphical positioning of the visible mark in the final displayed rendering of the sequences. A binary character property has also been introduced to refer to this class of marks as a whole: Prepended_Concatenation_Mark. Proper display of these marks requires specialized rendering support, as the shapes of the marks may adjust depending on the length of the following sequence of characters.

The use of subtending marks is most notably associated with the Arabic script. They typically occur before a sequence of digits and are then displayed with different styles of extended swashes underneath the digits. In Arabic, these marks often indicate whether the sequence of digits is to be interpreted as a number or a date, for example. Similar subtending marks are encoded for other scripts, including Syriac and Kaithi. (See *Section 9.2, Arabic*, *Section 9.3, Syriac*, and *Section 15.2, Kaithi* for a number of examples and further discussion.)

Combining Marks with Ligatures. According to *Section 3.6, Combination*, for a simple combining character sequence such as <i , ◌̂> , the nonspacing mark ◌̂ both *applies* to and *depends* on the base character *i*. If the *i* is preceded by a character that can ligate with it, additional considerations apply.

Figure 7-12 shows typical examples of the interaction of combining marks with ligatures. The sequence <f , i, ◌̂> is canonically equivalent to <f, î>. This implies that both sequences should be rendered identically, if possible. The precise way in which the sequence is rendered depends on whether the *f* and *i* of the first sequence ligate. If so, the result of applying ◌̂ should be the same as ligating an *f* with an *î*. The appearance depends on whatever typographical rules are established for this case, as illustrated in the first example of *Figure 7-12*. Note that the two characters *f* and *î* may not ligate, even if the sequence <f , i> does.

Figure 7-12. Interaction of Combining Marks with Ligatures

① $f + i + \hat{\circ} \equiv f + \hat{\imath} \rightarrow$ fî , fî , fî

② $f + \tilde{\circ} + i + \hat{\circ} \rightarrow$ f̃î , f̃î

③ $f + \hat{\circ} + i + \tilde{\circ} \rightarrow$ f̂ĩ , f̂ĩ

④ $f + \tilde{\circ} + i + \hat{\circ} \neq f + \hat{\circ} + i + \tilde{\circ}$

The second and third examples show that by default the sequence <f , ◌ , i , ◌> is visually distinguished from the sequence <f, ◌, i, ◌> by the relative placement of the accents. This is true whether or not the <f, ◌> and the <i, ◌> ligate. Example 4 shows that the two sequences are not canonically equivalent.

In some writing systems, established typographical rules further define the placement of combining marks with respect to ligatures. As long as the rendering correctly reflects the identity of the character sequence containing the marks, the Unicode Standard does not prescribe such fine typographical details.

Compatibility characters such as the *fi-ligature* are not canonically equivalent to the sequence of characters in their compatibility decompositions. Therefore, sequences like <fi-ligature, ◌> may legitimately differ in visual representation from <f, i, ◌>, just as the visual appearance of other compatibility characters may be different from that of the sequence of characters in their compatibility decompositions. By default, a compatibility character such as *fi-ligature* is treated as a single base glyph.

7.9.1 Combining Diacritical Marks: U+0300–U+036F

The combining diacritical marks in this block are intended for general use with any script. Diacritical marks specific to a particular script are encoded with that script. Diacritical marks that are primarily used with symbols are defined in the Combining Diacritical Marks for Symbols character block (U+20D0..U+20FF). For a detailed discussion of how multiple combining marks from this block interact when applied to a single base character, see "Multiple Combining Characters" in *Section 2.11, Combining Characters*.

Standards. The combining diacritical marks are derived from a variety of sources, including IPA, ISO 5426, and ISO 6937.

Underlining and Overlining. The characters U+0332 COMBINING LOW LINE, U+0333 COMBINING DOUBLE LOW LINE, U+0305 COMBINING OVERLINE, and U+033F COMBINING DOUBLE OVERLINE are intended to connect on the left and right. Thus, when used in combination, they could have the effect of continuous lines

above or below a sequence of characters. However, because of their interaction with other combining marks and other layout considerations such as intercharacter spacing, their use for underlining or overlining of text is discouraged in favor of using styled text.

7.9.2 Combining Diacritical Marks Extended: U+1AB0–U+1AFF

This block contains a set of combining diacritical marks used primarily in phonetic transcription for German dialectology. Certain characters are also used in other contexts.

Combining Parentheses. The combining diacritical marks U+1ABB COMBINING PARENTHESES ABOVE, U+1ABC COMBINING DOUBLE PARENTHESES ABOVE, and U+1ABD COMBINING PARENTHESES BELOW are used in German dialectology to indicate that the effect of a modifier on pronunciation is weakened. U+1AC5 COMBINING SQUARE BRACKETS ABOVE is used in Japanese transliteration to indicate that the accent mark contained within the square brackets is suppressed.

The positioning of these four combining parentheses diacritics deviates from the default stacking behavior of nonspacing marks. Instead of stacking vertically, they are placed side-by-side, surrounding the preceding diacritic above or below the base character. U+1ABB COMBINING PARENTHESES ABOVE and U+1ABC COMBINING DOUBLE PARENTHESES ABOVE are intended to be used with diacritics placed above, and U+1ABD COMBINING PARENTHESES BELOW is intended to be used with diacritics placed below. Correct positioning is illustrated in *Figure 7-13*. To indicate paired parentheses, U+1ABB..U+1ABD should be employed, rather than the single left and right combining parentheses marks located at U+1AC1..U+1AC4. The single combining parentheses are used in extended IPA to indicate initial or final voicing or devoicing, when appearing with a diacritic that marks voicing or devoicing.

Figure 7-13. Positioning of Combining Parentheses
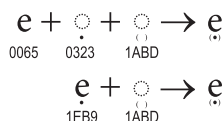
In contrast with the four combining paired parentheses diacritical marks above or below, which combine with other diacritics, U+1ABE COMBINING PARENTHESES OVERLAY is a regular enclosing mark, intended to surround a base character. The exact placement of the overlay U+1ABE with respect to a base character is not specified by the Unicode Standard, but may be adjusted for a particular base character as needed in fonts. For example, in the context of phonetic transcription for German dialectology, the combining character sequence <U+014B LATIN SMALL LETTER

ENG, U+1ABE COMBINING PARENTHESES OVERLAY> could be rendered with the parentheses placed lower to surround the descender of the letter *eng*.

7.9.3 Combining Diacritical Marks Supplement: U+1DC0– U+1DFF

This block is the supplement to the Combining Diacritical Marks block in the range U+0300..U+036F. It contains lesser-used combining diacritical marks.

U+1DC0 COMBINING DOTTED GRAVE ACCENT and U+1DC1 COMBINING DOTTED ACUTE ACCENT are marks occasionally seen in some Greek texts. They are variant representations of the accent combinations *dialytika varia* and *dialytika oxia*, respectively. They are, however, encoded separately because they cannot be reliably formed by regular stacking rules involving U+0308 COMBINING DIAERESIS and U+0300 COMBINING GRAVE ACCENT or U+0301 COMBINING ACUTE ACCENT.

U+1DC3 COMBINING SUSPENSION MARK is a combining mark specifically used in Glagolitic. It is not to be confused with a combining breve.

Typicon Kavyka Symbols. The Typicon is a liturgical book used in the Russian Orthodox Church and other Eastern Orthodox and Byzantine Catholic Churches. It contains liturgical instructions and other information for use in services. The Kavyka symbols were part of system used by the Russian liturgist Nikita Syrnikov that were derived from the archaic systems of Typicon symbols attributed to St. Gennadius, Archbishop of Novgorod, who served from 1484 to 1504 CE. Use of the Kavyka symbols is popular among members of the Yedinoverie community. *Table 7-5* describes their usage.

Table 7-5. Typicon Kavyka Symbols

Character	Use
1DF6 ◌̊ COMBINING KAVYKA ABOVE RIGHT	These indicate various rubrics for chanting the Beatitudes Troparia at the Divine Liturgy services.
1DF7 ◌̊ COMBINING KAVYKA ABOVE LEFT	
1DF8 ◌̊ COMBINING DOT ABOVE LEFT	This may occur with the combining *kavyki*.
1DF9 ◌̫ COMBINING WIDE INVERTED BRIDGE BELOW	This indicates that a double service has a common Exapostilarion.

7.9.4 Combining Diacritical Marks for Symbols: U+20D0–U+20FF

The combining marks in this block are generally applied to mathematical or technical symbols. They can be used to extend the range of the symbol set. For example, U+20D2 ◌⃒ COMBINING LONG VERTICAL LINE OVERLAY can be used to express negation, as shown in *Figure 7-14*. Its presentation may change in those circumstances— changing its length or slant, for example. That is, U+2261 ≡ IDENTICAL TO followed

by U+20D2 is equivalent to U+2262 ≢ NOT IDENTICAL TO. In this case, there is a precomposed form for the negated symbol. However, this statement does not always hold true, and U+20D2 can be used with other symbols to form the negation. For example, U+2258 CORRESPONDS TO followed by U+20D2 can be used to express *does not correspond to*, without requiring that a precomposed form be part of the Unicode Standard.

Figure 7-14. Use of Vertical Line Overlay for Negation

$$\equiv \; + \; \mid \; \rightarrow \; \not\equiv$$

2261   20D2

Other nonspacing characters are used in mathematical expressions. For example, a U+0304 COMBINING MACRON is commonly used in propositional logic to indicate logical negation.

Enclosing Marks. These nonspacing characters are supplied for compatibility with existing standards, allowing individual base characters to be enclosed in several ways. For example, U+2460 ① CIRCLED DIGIT ONE can be expressed as U+0031 DIGIT ONE "1" + U+20DD ◌⃝ COMBINING ENCLOSING CIRCLE. For additional examples, see *Figure 2-17*.

The combining enclosing marks surround their grapheme base and any intervening nonspacing marks. These marks are intended for application to free-standing symbols. See "Application of Combining Marks" in *Section 3.6, Combination*.

Users should be cautious when applying combining enclosing marks to other than free-standing symbols—for example, when using a combining enclosing circle to apply to a letter or a digit. Most implementations assume that application of any nonspacing mark will not change the character properties of a base character. This means that even though the intent might be to create a circled symbol (General_Category = So), most software will continue to treat the base character as an alphabetic letter or a numeric digit. Note that there is no *canonical* equivalence between a symbolic character such as U+24B6 CIRCLED LATIN CAPITAL LETTER A and the sequence <U+0041 LATIN CAPITAL LETTER A, U+20DD COMBINING ENCLOSING CIRCLE>, partly because of this difference in treatment of properties.

7.9.5 Combining Half Marks: U+FE20–U+FE2F

This block consists of a number of presentation form (glyph) encodings that may be used to visually encode certain combining marks that apply to multiple base letter-forms. These characters are intended to facilitate the support of such marks in legacy implementations.

Unlike other compatibility characters, these half marks do not correspond directly to a single character or a sequence of characters; rather, a discontiguous sequence of the combining half marks corresponds to a single combining mark, as depicted in *Figure 7-15*. The preferred forms are the double diacritics, such as U+0360 COMBINING DOUBLE TILDE. See the earlier discussion of "Diacritics Positioned Above Two Base Characters."

Figure 7-15. Double Diacritics and Half Marks

Using Combining Half Marks

n + ◌͢ + g + ◌ͣ → n͡g
006E   FE22   0067   FE23

Using Double Diacritics

n + ◌͠ + g → n͠g
006E   0360   0067

This block also contains half marks for macrons and conjoining macrons, both above and below. These marks can be used in combinations on successive letters to support particular styles of supralineation or sublineation in some historic scripts. See, for example, *Section 7.3, Coptic*. However, lines which extend across more than two letters may be better rendered if expressed in terms of explicit text styles, rather than by a series of combining half marks, applied one letter at a time in the plain text sequence.

7.9.6 Combining Marks in Other Blocks

In addition to the blocks of characters in the standard specifically set aside for combining marks, many combining marks are associated with particular scripts or occasionally with groups of scripts. Thus the Arabic block contains a large collection of combining marks used to indicate vowelling of Arabic text as well as another collection of combining marks used in annotation of Quranic text. Such marks are mostly intended for use with the Arabic script, but in some instances other scripts, such as Syriac, may use them as well.

Nearly every Indic script has its own collection of combining marks, notably including sets of combining marks to represent dependent vowels, or *matras*.

In some instances a combining mark encoded specifically for a given script, and located in the code chart for that script, may look very similar to a diacritical mark from one of the blocks dedicated to generic combining marks. In such cases, a variety of reasons, including rendering behavior in context or patterning considerations, may have led to separate encoding. The general principle is that if a correctly identified

script-specific combining mark of the appropriate shape is available, that character is intended for use with that script, in lieu of a generic combining mark that might look similar. If a combining mark of the appropriate shape is not available in the relevant script block or blocks, then one should make use of whichever generic combining mark best suits the intended purpose.

For example, in representing Syriac text, to indicate a dot above a letter that was identified as a *qushshaya*, one would use U+0741 SYRIAC QUSHSHAYA rather than the generic U+0307 COMBINING DOT ABOVE. When attempting to represent a *hamza* above a Syriac letter, one would use U+0654 ARABIC HAMZA ABOVE, which is intended for both Arabic and Syriac, because there is no specifically Syriac *hamza* combining mark. However, if marking up Syriac text with diacritics such as a macron to indicate length or some other feature, one would then make use of U+0304 COMBINING MACRON from the generic block of combining diacritical marks.

Chapter 8

Europe-II
Ancient and Other Scripts

This chapter describes ancient scripts of Europe, as well as other historic and limited-use scripts of Europe not covered in *Chapter 7, Europe-I*. This includes the various ancient Mediterranean scripts, other early alphabets and sets of runes, some poorly attested historic scripts of paleographic interest, and more recently devised constructed scripts with significant usage.

Unicode encodes a number of ancient scripts, which have not been in normal use for a millennium or more, as well as historic scripts, whose usage ended in recent centuries. Although they are no longer used to write living languages, documents and inscriptions using these scripts exist, both for extinct languages and for precursors of modern languages. The primary user communities for these scripts are scholars interested in studying the scripts and the languages written in them. Some of the historic scripts are related to each other as well as to modern alphabets.

The Linear A script is an ancient writing system used from approximately 1700–1450 BCE on and around the island of Crete. The script contains more than ninety signs in regular use and a host of logograms. Surviving examples are inscribed on clay tablets, stone tables, and metals. The language of the inscriptions has not yet been deciphered.

Both Linear B and Cypriot are syllabaries that were used to write Greek. Linear B is the older of the two scripts, and there are some similarities between a few of the characters that may not be accidental. Cypriot may descend from Cypro-Minoan, which in turn may descend from Linear B.

Cypro-Minoan is an undeciphered script from the late Bronze Age (circa 1550-1050 BCE) found on objects from the island of Cyprus, the ancient cities of Ugarit (modern-day Ras Shamra, Syria) and Tiryns, Greece. It is a syllabic script.

The ancient Anatolian alphabets Lycian, Carian, and Lydian all date from the first millennium BCE, and were used to write various ancient Indo-European languages of western and southwestern Anatolia. All are closely related to the Greek script.

Old Italic was derived from Greek and was used to write Etruscan and other languages in Italy. It was borrowed by the Romans and is the immediate ancestor of

the Latin script now used worldwide. One of the Old Italic alphabets of northern Italy may have influenced the development of the Runic script, which has a distinct angular appearance owing to its use in carving inscriptions in stone and wood.

Old Hungarian is another historical runiform script, used to write the Hungarian language in Central Europe. In recent decades it has undergone a significant revival in Hungary. It has developed casing, and is now used with modern typography to print significant amounts of material in the modern Hungarian language. It is laid out from right to left.

The Ogham script is indigenous to Ireland. While its originators may have been aware of the Latin or Greek scripts, it seems clear that the sound values of Ogham letters were suited to the phonology of a form of Primitive Irish.

The Gothic script, like Cyrillic, was developed on the basis of Greek at a much later date than Old Italic.

Elbasan, Vithkuqi, and Todhri are all simple, left-to-right alphabetic scripts, used historically to write Albanian. Elbasan was invented in the middle of the eighteenth-century. It is named after the city where it originated. Vithkuqi was invented by Naum P. Veqilharxhi, and is named for the town where it was created in the nineteenth century. It is experiencing some modern revivalist efforts in artistic and cultural uses. The Todhri alphabet is another historical script used to write the Albanian language in the central Albanian region that is now designated Elbasan County. It was used in the eighteenth and nineteenth centuries and perhaps sporadically into the twentieth century.

Caucasian Albanian is a poorly attested simple alphabetic script that dates from the early fifth century. The text of its attestation is related to the modern Udi language.

Old Permic is a simple alphabetic script devised in the fourteenth century to write the Uralic languages Komi and Komi-Permyak. Its use for Komi extended into the seventeenth century.

Shavian is a phonemic alphabet invented in the 1950s to write English. It was used to publish one book in 1962, but remains of some current interest.

8.1 Linear A

8.1.1 Linear A: U+10600–U+1077F

The Linear A script was used from approximately 1700–1450 BCE. It was mainly used on the island of Crete and surrounding areas to write a language which has not yet been identified. Unlike the later Linear B, which was used to write an early form of Greek, Linear A appears on a variety of media, such as clay tablets, stone offering tables, gold and silver hair pins, and pots.

Encoding. The repertoire of characters in the Unicode encoding of the Linear A script is broadly based on the GORILA catalog by Godart and Olivier (1976–1985), which is the basic set of signs used in decipherment efforts. All simple signs in that catalog are encoded as single characters. Composite signs consisting of vertically stacked parts or touching pieces are also encoded as single characters. Composite signs in the catalog which consist of side-by-side pieces that are not touching are treated as digraphs; the parts are individually encoded as characters, but the composite sign is not separately encoded.

Structure. Linear A contains more than ninety syllabic signs in regular use and a host of logograms. Some Linear A signs are also found in Linear B, although about 80% of the logograms in Linear A do not appear in Linear B.

Character Names. The Linear A character names are based on the GORILA catalog numbers.

Directionality. Linear A was written from left to right, though occasionally it appears right to left and, rarely, boustrophedon.

Numbers. Numbers in Linear A inscriptions are represented by characters in the Aegean Numbers block. Numbers are usually arranged in sets of five or fewer that are stacked vertically. The largest number recorded is 3,000. Linear A seems to use a series of unit fractions. Seven fractions are regularly used and are included in the Linear A block.

8.2 Linear B

8.2.1 Linear B Syllabary: U+10000–U+1007F

The Linear B script is a syllabic writing system that was used on the island of Crete and parts of the nearby mainland to write the oldest recorded variety of the Greek language. Linear B clay tablets predate Homeric Greek by some 700 years; the latest tablets date from the mid- to late thirteenth century BCE. Major archaeological sites include Knossos, first uncovered about 1900 by Sir Arthur Evans, and a major site near Pylos. The majority of currently known inscriptions are inventories of commodities and accounting records.

Early attempts to decipher the script failed until Michael Ventris, an architect and amateur decipherer, came to the realization that the language might be Greek and not, as previously thought, a completely unknown language. Ventris worked together with John Chadwick, and decipherment proceeded quickly. The two published a joint paper in 1953.

Linear B was written from left to right with no nonspacing marks. The script mainly consists of phonetic signs representing the combination of a consonant and a vowel. There are about 60 known phonetic signs, in addition to a few signs that seem to be mainly free variants (also known as Chadwick's optional signs), a few unidentified signs, numerals, and a number of ideographic signs, which were used mainly as counters for commodities. Some ligatures formed from combinations of syllables were apparently used as well. Chadwick gives several examples of these ligatures, the most common of which are included in the Unicode Standard. Other ligatures are the responsibility of the rendering system.

Standards. The catalog numbers used in the Unicode character names for Linear B syllables are based on the Wingspread Convention, as documented in Bennett (1964). The letter "B" is prepended arbitrarily, so that name parts will not start with a digit, thus conforming to ISO/IEC 10646 naming rules. The same naming conventions, using catalog numbers based on the Wingspread Convention, are used for Linear B ideograms.

8.2.2 Linear B Ideograms: U+10080–U+100FF

The Linear B Ideograms block contains the list of Linear B signs known to constitute ideograms (logographs), rather than syllables. When generally agreed upon, the names include the meaning associated with them—for example, U+10080 𐂀 LINEAR B IDEOGRAM B100 MAN. In other instances, the names of the ideograms simply carry their catalog number.

8.2.3 Aegean Numbers: U+10100–U+1013F

The signs used to denote Aegean whole numbers (U+10107..U+10133) derive from the non-Greek Linear A script. The signs are used in Linear B. The Cypriot syllabary appears to use the same system, as evidenced by the fact that the lower digitsappear in extant texts. For measurements of agricultural and industrial products, Linear B uses three series of signs: liquid measures, dry measures, and weights. No set of signs for linear measurement has been found yet. Liquid and dry measures share the same symbols for the two smaller subunits; the system of weights retains its own unique subunits. Though several of the signs originate in Linear A, the measuring system of Linear B differs from that of Linear A. Linear B relies on units and subunits, much like the imperial "quart," "pint," and "cup," whereas Linear A uses whole numbers and fractions. The absolute values of the measurements have not yet been completely agreed upon.

8.3 Cypriot Syllabary

8.3.1 Cypriot Syllabary: U+10800–U+1083F

The Cypriot syllabary was used to write the Cypriot dialect of Greek from about 800 to 200 BCE. It is related to both Linear B and Cypro-Minoan, a script used for a language that has not yet been identified. Interpretation has been aided by the fact that, as use of the Cypriot syllabary died out, inscriptions were carved using both the Greek alphabet and the Cypriot syllabary. Unlike Linear B and Cypro-Minoan, the Cypriot syllabary was usually written from right to left, and accordingly the characters in this script have strong right-to-left directionality.

Word breaks can be indicated by spaces or by separating punctuation, although separating punctuation is also used between larger word groups.

Although both Linear B and the Cypriot syllabary were used to write Greek dialects, Linear B has a more highly abbreviated spelling. Structurally, the Cypriot syllabary consists of combinations of up to 12 initial consonants and 5 different vowels. Long and short vowels are not distinguished. The Cypriot syllabary distinguishes among a different set of initial consonants than Linear B; for example, unlike Linear B, Cypriot maintained a distinction between [l] and [r], though not between [d] and [t], as shown in *Table 8-1*. Not all of the 60 possible consonant-vowel combinations are represented. As is the case for Linear B, the Cypriot syllabary is well understood and documented.

Table 8-1. Similar Characters in Linear B and Cypriot

Linear B		Cypriot	
da	⊢	*ta*	⊢
na	⫪	*na*	⊤
pa	ǂ	*pa*	ǂ
ro	⊹	*lo*	⊹
se	⸗	*se*	⸗
ti	⋀	*ti*	↑
to	⊤	*to*	ℾ

For Aegean numbers, see the subsection "Aegean Numbers: U+10100–U+1013F" in *Section 8.2, Linear B*.

8.4 Cypro-Minoan

8.4.1 Cypro-Minoan: U+12F90–U+12FFF

Cypro-Minoan is an undeciphered script found on approximately 250 objects from the island of Cyprus, the ancient cities of Ugarit (modern-day Ras Shamra, Syria) and Tiryns, Greece. The script dates to the late Bronze Age (circa 1550-1050 BCE). The name "Cypro-Minoan" was coined by Arthur Evans in 1909 because he believed Cypro-Minoan derived from the scripts of Minoan Crete.

Researchers have tentatively classified Cypro-Minoan into four categories, termed CM0, CM1, CM2, and CM3, based on temporal and geographical criteria. The repertoire in the Unicode Standard covers characters from the CM1, CM2, and CM3 groups, but does not cover CM0; it is largely based on Olivier 2007.

Structure. Cypro-Minoan is a syllabic script and has been encoded with left-to-right directionality.

Names. The character names are based on Olivier 2007.

Glyphs. The glyphs in the code charts generally follow the CM1 forms, but if no CM1 form exists, a CM2 or CM3 form is used. The glyphs follow Olivier 2007 generally, except for U+12F9C CYPRO-MINOAN SIGN CM013, which has been modified based on recent research. The code chart normalizes the glyphs into a more linear style.

Punctuation. A few Cypro-Minoan punctuation marks have been identified. Two script-specific signs are encoded: U+12FF1 CYPRO-MINOAN SIGN CM301 and U+12FF2 CYPRO-MINOAN SIGN CM302. Two other marks have been unified with two punctuation characters in the Aegean Numbers block: U+10100 AEGEAN WORD SEPARATOR LINE and U+10101 AEGEAN WORD SEPARATOR DOT.

Numbers. Numbers in Cypro-Minoan are known, but poorly attested. Users may choose to employ characters from the Aegean numbers block for Cypro-Minoan, but the exact relationship between the Cypro-Minoan and Aegean numbers remains uncertain.

8.5 Ancient Anatolian Alphabets

8.5.1 Lycian: U+10280–U+1029F

Carian: U+102A0–U+102DF

Lydian: U+10920–U+1093F

The Anatolian scripts described in this section all date from the first millennium BCE, and were used to write various ancient Indo-European languages of western and southwestern Anatolia (now Turkey). All are closely related to the Greek script and are probably adaptations of it. Additional letters for some sounds not found in Greek were probably either invented or drawn from other sources. However, development parallel to, but independent of, the Greek script cannot be ruled out, particularly in the case of Carian.

Lycian. Lycian was used from around 500 BCE to about 200 BCE. The term "Lycian" is now used in place of "Lycian A" (a dialect of Lycian, attested in two texts in Anatolia, is called "Lycian B", or "Milyan", and dates to the first millennium BCE). The Lycian script appears on some 150 stone inscriptions, more than 200 coins, and a few other objects.

Lycian is a simple alphabetic script of 29 letters, written from left to right, with frequent use of word dividers. The recommended word divider is U+205A TWO DOT PUNCTUATION. *Scriptio continua* (a writing style without spaces or punctuation) also occurs. In modern editions U+0020 SPACE is sometimes used to separate words.

Carian. The Carian script is used to write the Carian language, and dates from the first millennium BCE. While a few texts have been found in Caria, most of the written evidence comes from Carian communities in Egypt, where they served as mercenaries. The repertoire of the Carian texts is well established. Unlike Lycian and Lydian, Carian does not use a single standardized script, but rather shows regional variation in the repertoire of signs used and their form. Although some of the values of the Carian letters remain unknown or in dispute, their distinction from other letters is not. The Unicode encoding is based on the standard "Masson set" catalog of 45 characters, plus 4 recently-identified additions. Some of the characters are considered to be variants of others—and this is reflected in their names—but are separately encoded for scholarly use in discussions of decipherment.

The primary direction of writing is left-to-right in texts from Caria, but right-to-left in Egyptian Carian texts. However, both directions occur in the latter, and left-to-right is favored for modern scholarly usage. Carian is encoded in Unicode with left-to-right directionality. Word dividers are not regularly employed; *scriptio continua* is common. Word dividers which are attested are U+00B7 MIDDLE DOT (or U+2E31

WORD SEPARATOR MIDDLE DOT), U+205A TWO DOT PUNCTUATION, and U+205D TRICOLON. In modern editions U+0020 SPACE may be found.

Lydian. While Lydian is attested from inscriptions and coins dating from the end of the eighth century (or beginning of the seventh) until the third century BCE, the longer well-preserved inscriptions date to the fifth and fourth centuries BCE.

Lydian is a simple alphabetic script of 26 letters. The vast majority of Lydian texts have right-to-left directionality (the default direction); a very few texts are left-to-right and one is boustrophedon. Most Lydian texts use U+0020 SPACE as a word divider. Rare examples have been found which use *scriptio continua* or which use dots to separate the words. In the latter case, U+003A COLON and U+00B7 MIDDLE DOT (or U+2E31 WORD SEPARATOR MIDDLE DOT) can be used to represent the dots. U+1093F LYDIAN TRIANGULAR MARK is thought to indicate quotations, and is mirrored according to text directionality.

8.6 Old Italic

8.6.1 Old Italic: U+10300–U+1032F

The Old Italic script is used to represent a number of related historical alphabets located on the Italian peninsula. Some of these were used for non-Indo-European languages (Etruscan, Raetic, and probably North Picene), and some for various Indo-European languages belonging to the Italic branch (Faliscan and members of the Sabellian group, including Oscan, Umbrian, and South Picene) the Celtic branch (Cisalpine Celtic), and the Venetic branch. The ultimate source for the alphabets in ancient Italy is Euboean Greek used at Ischia and Cumae in the bay of Naples in the eighth century BCE. Unfortunately, no Greek abecedaries from southern Italy have survived. The native alphabets of Faliscan, Oscan, Umbrian, North Picene, South Picene, Venetic, and Cisalpine Celtic all derive from an Etruscan form of the alphabet. Raetic, or another Old Italic alphabet of northern Italy, may have influenced the historical development of Runic. (See *Section 8.7, Runic*.)

There are some 10,000 inscriptions in Etruscan. By the time of the earliest Etruscan inscriptions, circa 700 BCE, local distinctions are already found in the use of the alphabet. Three major stylistic divisions are identified: the Northern, Southern, and Caere/Veii. Use of Etruscan can be divided into two stages, owing largely to the phonological changes that occurred: the "archaic Etruscan alphabet," used from the seventh to the fifth centuries BCE, and the "neo-Etruscan alphabet," used from the fourth to the first centuries BCE. Glyphs for eight of the letters differ between the two periods; additionally, neo-Etruscan abandoned the letters KA, KU, and EKS.

The unification of these alphabets into a single Old Italic script requires language-specific fonts because the glyphs most commonly used may differ somewhat depending on the language being represented.

Most of the languages have added characters to the common repertoire: Etruscan and Faliscan add LETTER EF; Oscan adds LETTER EF, LETTER II, and LETTER UU; Umbrian adds LETTER EF, LETTER ERS, and LETTER CHE; North Picene adds LETTER UU; South Picene adds LETTER II, LETTER UU, and LETTER ESS; Venetic adds LETTER YE; and Raetic adds NORTHERN TSE and SOUTHERN TSE.

The Latin script itself derives from a south Etruscan model, probably from Caere or Veii, around the mid-seventh century BCE or a bit earlier. However, because there are significant differences between Latin and Faliscan of the seventh and sixth centuries BCE in terms of formal differences (glyph shapes, directionality) and differences in the repertoire of letters used, this warrants a distinctive character block. Fonts for early Latin should use the *uppercase* code positions U+0041..U+005A.

Character names assigned to the Old Italic block are unattested but have been reconstructed according to the analysis made by Sampson (1985). While the Greek character names (ALPHA, BETA, GAMMA, and so on) were borrowed directly from the Phoenician names (modified to Greek phonology), the Etruscans are thought to have abandoned the Greek names in favor of a phonetically based nomenclature, where stops were pronounced with a following -e sound, and liquids and sibilants (which can be pronounced more or less on their own) were pronounced with a leading e-sound (so [k], [d] became [ke:], [de:], while [l], [m] became [el], [em]). It is these names, according to Sampson, which were borrowed by the Romans when they took their script from the Etruscans.

Directionality. Most Etruscan texts from the seventh to six centuries BCE were written from right to left, but writing left to right was not uncommon, and is found in approximately ten percent of the texts from this period. From the fifth to the first centuries BCE, writing right to left was the standard, and left-to-right directionality was extremely rare. The other local varieties of Old Italic also generally have right-to-left directionality. Boustrophedon appears rarely, and not especially early (for instance, the Forum inscription dates to 550–500 BCE). Despite this, for reasons of implementation simplicity, many scholars prefer left-to-right presentation of texts, as this is also their practice when transcribing the texts into Latin script. Accordingly, the Old Italic script has a default directionality of strong left-to-right in this standard. If the default directionality of the script is overridden to produce a right-to-left presentation, the glyphs in Old Italic fonts should also be mirrored from the representative glyphs shown in the code charts. This kind of behavior is not uncommon in archaic scripts; for example, archaic Greek letters may be mirrored when written from right to left in boustrophedon.

Punctuation. The earliest inscriptions are written with no space between words in what is called *scriptio continua*. There are numerous Etruscan inscriptions with dots separating word forms, attested as early as the second quarter of the seventh century BCE. This punctuation is sometimes, but only rarely, used to mark certain types of syllables and not to separate words. From the sixth century BCE, words were often separated by one, two, or three dots spaced vertically above each other.

Numerals. Etruscan numerals are not well attested in the available materials, but are employed in the same fashion as Roman numerals. Several additional numerals are attested, but as their use is at present uncertain, they are not yet encoded in the Unicode Standard.

Glyphs. The default glyphs in the code charts are based on the most common shapes found for each letter. Most of these are similar to the Marsiliana abecedary (mid-seventh century BCE). Note that the phonetic values for U+10317 OLD ITALIC LETTER EKS [ks] and U+10319 OLD ITALIC LETTER KHE [k^h] show the influence of western, Euboean Greek; eastern Greek has U+03A7 GREEK CAPITAL LETTER CHI [k^h] and U+03A8 GREEK CAPITAL LETTER PSI [ps] instead.

The geographic distribution of the Old Italic script is shown in *Figure 8-1*. In the figure, the approximate distribution of the ancient languages that used Old Italic alphabets is shown in white. Areas for the ancient languages that used other scripts are shown in gray, and the labels for those languages are shown in italics. In particular, note that the ancient Greek colonies of the southern Italian and Sicilian coasts used the Greek script proper. Rome, of course, is shown in gray, because Latin was written with the Latin alphabet, now encoded in the Latin script.

Figure 8-1. Distribution of Old Italic
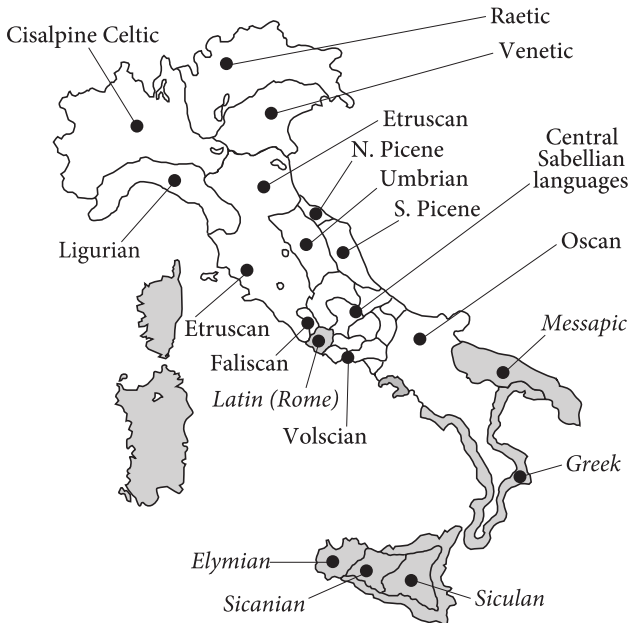
8.7 Runic

8.7.1 Runic: U+16A0–U+16FF

The Runic script was historically used to write the languages of the early and medieval societies in the German, Scandinavian, and Anglo-Saxon areas. Use of the Runic script in various forms covers a period from the first century to the nineteenth century. Some 6,000 Runic inscriptions are known. They form an indispensable source of information about the development of the Germanic languages.

The Runic script is an historical script, whose most important use today is in scholarly and popular works about the old Runic inscriptions and their interpretation. The Runic script illustrates many technical problems that are typical for this kind of script. Unlike many other scripts in the Unicode Standard, which predominantly serve the needs of the modern user community—with occasional extensions for historic forms—the encoding of the Runic script attempts to suit the needs of texts from different periods of time and from distinct societies that had little contact with one another.

The Runic Alphabet. Present-day knowledge about runes is incomplete. The set of graphemically distinct units shows greater variation in its graphical shapes than most modern scripts. The Runic alphabet changed several times during its history, both in the number and the shapes of the letters contained in it. The shapes of most runes can be related to some Latin capital letter, but not necessarily to a letter representing the same sound. The most conspicuous difference between the Latin and the Runic alphabets is the order of the letters.

The Runic alphabet is known as the *futhark* from the name of its first six letters. The original *old futhark* contained 24 runes:

ᚠ	ᚢ	ᚦ	ᚨ	ᚱ	ᚲ	ᚷ	ᚹ		ᚺ	ᚾ	ᛁ	ᛃ	ᛄ	ᛈ	ᛉ	ᛊ		ᛏ	ᛒ	ᛖ	ᛗ	ᛚ	ᛜ	ᛞ	ᛟ
They are usually transliterated in this way:																									
f	u	þ	a	r	k	g	w		h	n	i	j	ï	p	z	s		t	b	e	m	l	ŋ	d	o

In England and Friesland, seven more runes were added from the fifth to the ninth century.

In the Scandinavian countries, the *futhark* changed in a different way; in the eighth century, the simplified younger *futhark* appeared. It consists of only 16 runes, some of which are used in two different forms. The long-branch form is shown here:

ᚠ	ᚢ	ᚦ	ᚭ	ᚱ	ᚴ		ᚼ	ᚾ	ᛁ	ᛅ	ᛋ		ᛏ	ᛒ	ᛘ	ᛚ	ᛦ
f	u	þ	o	r	k		h	n	i	a	s		t	b	m	l	ʀ

The use of runes continued in Scandinavia during the Middle Ages. During that time, the *futhark* was influenced by the Latin alphabet and new runes were invented so that there was full correspondence with the Latin letters.

Direction. Like other early writing systems, runes could be written either from left to right or from right to left, or moving first in one direction and then the other (*boustrophedon*), or following the outlines of the inscribed object. At times, characters appear in mirror image, or upside down, or both. In modern scholarly literature, Runic is written from left to right. Therefore, the letters of the Runic script have a default directionality of strong left-to-right in this standard.

Representative Glyphs. The known inscriptions can include considerable variations of shape for a given rune, sometimes to the point where the nonspecialist will mistake the shape for a different rune. There is no dominant main form for some runes, particularly for many runes added in the Anglo-Friesian and medieval Nordic systems. When transcribing a Runic inscription into its Unicode-encoded form, one cannot rely on the idealized *representative glyph* shape in the character charts alone. One must take into account to which of the four Runic systems an inscription belongs and be knowledgeable about the permitted form variations within each system. The representative glyphs were chosen to provide an image that distinguishes each rune visually from all other runes in the same system. For actual use, it might be advisable to use a separate font for each Runic system. Of particular note is the fact that the glyph for U+16C4 ᛄ RUNIC LETTER GER is actually a rare form, as the more common form is already used for U+16E1 ᛡ RUNIC LETTER IOR.

Unifications. When a rune in an earlier writing system evolved into several different runes in a later system, the unification of the earlier rune with one of the later runes was based on similarity in graphic form rather than similarity in sound value. In cases where a substantial change in the typical graphical form has occurred, though the historical continuity is undisputed, unification has not been attempted. When runes from different writing systems have the same graphic form but different origins and denote different sounds, they have been coded as separate characters.

Long-Branch and Short-Twig. Two sharply different graphic forms, the *long-branch* and the *short-twig* form, were used for 9 of the 16 Viking Age Nordic runes. Although only one form is used in a given inscription, there are runologically important exceptions. In some cases, the two forms were used to convey different meanings in later use in the medieval system. Therefore the two forms have been separated in the Unicode Standard.

Staveless Runes. Staveless runes are a third form of the Viking Age Nordic runes, a kind of Runic shorthand. The number of known inscriptions is small and the graphic forms of many of the runes show great variability between inscriptions. For this reason, staveless runes have been unified with the corresponding Viking Age Nordic runes. The corresponding Viking Age Nordic runes must be used to encode these

characters—specifically the short-twig characters, where both short-twig and long-branch characters exist.

Punctuation Marks. The wide variety of Runic punctuation marks has been reduced to three distinct characters based on simple aspects of their graphical form, as very little is known about any difference in intended meaning between marks that look different. Any other punctuation marks have been unified with shared punctuation marks elsewhere in the Unicode Standard.

Golden Numbers. Runes were used as symbols for Sunday letters and golden numbers on calendar staves used in Scandinavia during the Middle Ages. To complete the number series 1–19, three more calendar runes were added. They are included after the punctuation marks.

Encoding. The order of the Runic characters follows the traditional *futhark* order, with variants and derived runes being inserted directly after the corresponding ancestor.

Runic character names are based as much as possible on the sometimes several traditional names for each rune, often with the Latin transliteration at the end of the name.

8.8 Old Hungarian

8.8.1 Old Hungarian: U+10C80–U+10CFF

The Old Hungarian script is a runiform script that is used to write the Hungarian language. Old Hungarian is mentioned in a written account of the late 13th century and has been found on short stone-carved inscriptions. The script was probably developed and in use earlier. Modern use has increased dramatically in the last two decades, with some uses being simply decorative. There are also currently publications of books, magazines, and teaching materials.

Structure. Old Hungarian is an alphabetic script. The consonants traditionally bore an inherent vowel. Vowel signs were only explicitly written in final position, where vowels were long, and for disambiguation. In later phases of script usage, all vowels were written explicitly. The script is rendered linearly, but traditionally used a large set of ligatures.

Casing is not part of the traditional Old Hungarian script. However, modern practice has introduced casing into many publications. Uppercase letters appear as larger size variants of lowercase letters.

Directionality. The primary direction of writing is right-to-left both in historical sources and in modern use. Conformant implementations of Old Hungarian script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Punctuation and Numbers. Traditional texts separate words with spaces or with one, two, or four dots. Modern users punctuate Old Hungarian with U+0020 SPACE, U+2E41 REVERSED COMMA, and U+2E42 DOUBLE LOW-REVERSED-9 QUOTATION MARK, with some use of U+2E31 WORD SEPARATOR MIDDLE DOT, U+205A TWO DOT PUNCTUATION, U+205D TRICOLON, and U+205E VERTICAL FOUR DOTS as well.

Old Hungarian numbers have their origin in a tally system which was widely used throughout Hungary until the nineteenth century. Since the twentieth century, these numbers have been used regularly with Old Hungarian. The numbers are built up from elements in a multiplicative-additive system. Old Hungarian numbers are encoded at U+10CFA..U+10CFF.

8.9 Gothic

8.9.1 Gothic: U+10330–U+1034F

The Gothic script was devised in the fourth century by the Gothic bishop, Wulfila (311–383 CE), to provide his people with a written language and a means of reading his translation of the Bible. Written Gothic materials are largely restricted to fragments of Wulfila's translation of the Bible; these fragments are of considerable importance in New Testament textual studies. The chief manuscript, kept at Uppsala, is the Codex Argenteus or "the Silver Book," which is partly written in gold on purple parchment. Gothic is an East Germanic language; this branch of Germanic has died out and thus the Gothic texts are of great importance in historical and comparative linguistics. Wulfila appears to have used the Greek script as a source for the Gothic, as can be seen from the basic alphabetical order. Some of the character shapes suggest Runic or Latin influence, but this is apparently coincidental.

Diacritics. The tenth letter U+10339 GOTHIC LETTER EIS is used with U+0308 COMBINING DIAERESIS when word-initial, when syllable-initial after a vowel, and in compounds with a verb as second member as shown below:

SYG ГΛΜGΛΙϕ ϊST ϊN GSΛϊϊN ΠRΛΠFGTΛΠ
swe gameliþ ïst ïn esaïïn praufetau
"as is written in Isaiah the prophet"

Numerals. Gothic letters, like those of other early Western alphabets, can be used as numbers; two of the characters have only a numeric value and are not used alphabetically. To indicate numeric use of a letter, it is either flanked on both sides by U+00B7 MIDDLE DOT or followed by both U+0304 COMBINING MACRON and U+0331 COMBINING MACRON BELOW, as shown in the following example:

•G• or Ḡ means "5"

Punctuation. Gothic manuscripts are written with no space between words in what is called *scriptio continua*. Sentences and major phrases are often separated by U+0020 SPACE, U+00B7 MIDDLE DOT, or U+003A COLON.

8.10 Elbasan

8.10.1 Elbasan: U+10500–U+1052F

The earliest alphabet devised for the Albanian language was created around 1750 for the Elbasan Gospel manuscript, which is the only known example of the script. This manuscript, preserved at the State Archives in Tirana, records the earliest-known Albanian-language text in an original alphabet. Most of the letters in the Elbasan alphabet seem to be new creations, although some of their shapes may have been influenced by Greek and Cyrillic.

Structure. Elbasan is a simple alphabetic script written from left to right horizontally. The alphabet consists of forty letters.

Three characters have an inherent diacritical dot: U+10505 ⊥ ELBASAN LETTER NDE is used to indicate a pre-nasalized U+10504 ⊥ /d/; U+10511 ⋏ ELBASAN LETTER LLE is used to indicate a geminate U+10510 ⋏ /l/; U+1051A ⊥ ELBASAN LETTER RRE is used to indicate a geminate U+10519 ⊥ /r/. In many cases the dot on *nde* is written like a small *ne*. In one instance in the manuscript *gje* is written with a dot above to indicate prenasalized /ɟ/.

Two different letters are used for /n/: U+10513 N ELBASAN LETTER NE is used generally, and U+10514 ⋪ ELBASAN LETTER NA is typically used in prenasalized position as in ⋪ঠ /ng/ and ⋪φ /nɟ/. Two letters, which are rare and appear in Greek loanwords, are used for /γ/, U+10525 F ELBASAN LETTER GHE and U+10526 ∟ ELBASAN LETTER GHAMMA.

Accents and Other Marks. The Elbasan manuscript contains breathing accents, similar to those used in Greek. Those accents do not appear regularly in the orthography and have not been fully analyzed yet. Raised vertical marks also appear in the manuscript, but are not specific to the script. Generic combining characters from the Combining Diacritical Marks block can be used to render these accents and other marks.

Names. The names used for the characters in the Elbasan block are based on those of the modern Albanian alphabet.

Numerals and Punctuation. There are no script-specific numerals or punctuation marks. A separating dot and spaces appear in the Elbasan manuscript, and may be rendered with U+00B7 MIDDLE DOT and U+0020 SPACE, respectively. For numerals, a Greek-like system of letter and combining overline is in use. Overlines also appear above certain letters in abbreviations, such as $\overline{\varsigma\nabla}$ to indicate *Zot* (Lord). The overlines in numerals and abbreviations can be represented with U+0305 COMBINING OVERLINE.

8.11 Caucasian Albanian

8.11.1 Caucasian Albanian: U+10530–U+1056F

The Caucasian Albanian script was identified as a unique script in 1937 on the basis of an alphabet list in an Armenian manuscript in the Matenadaran collection in Yerevan and confirmed by a few inscriptions on artifacts excavated in northwest Azerbaijan around 1950. In the 1990s two palimpsest manuscripts containing the Caucasian Albanian script were discovered in St. Catherine's Monastery on Mount Sinai. These undated manuscripts appear to have been written during the seventh century CE. The palimpsests were deciphered and the Caucasian Albanian language and script was determined to be closely related to, if not an ancestor of, the present-day Udi language.

Structure. Caucasian Albanian is a simple alphabetic script written from left to right horizontally. Spaces are not used to separate words in the manuscript, though modern editions use spaces for the better legibility.

Abbreviations. An abbreviatory convention occurs, using a line above spanning two letters. This line above has a titlo-appearance, with small fillets at the ends of the strokes. This convention is similar to that seen in Coptic. For Caucasian Albanian, use of U+035E COMBINING DOUBLE MACRON is recommended to represent such abbreviations, with the font design dealing with the swash ends of the line, for styles that require it.

Numerals. Script-specific numerals are not known. Letters used as numbers are marked with a line above and/or below the letter, so Ƃ̄ or Ƃ̱ or Ƃ̱̄ = 2. When more than two or three letters are associated with a numeric mark, a continuous line is drawn above or below all of them. These lines above and/or below can be represented with U+0304 COMBINING MACRON, U+0331 COMBINING MACRON BELOW, or with various combinations of combining half macrons and conjoining macrons from the Combining Half Marks block (U+FE20..U+FE2F), as needed. (See the discussion of supralineation in *Section 7.3, Coptic*.)

Punctuation. One special mark, U+1056F CAUCASIAN ALBANIAN CITATION MARK, is used to indicate text that is a citation from the psalms.

8.12 Vithkuqi

8.12.1 Vithkuqi: U+10570–U+105BF

Vithkuqi, a historical script for Albanian, was invented by Naum P. Veqilharxhi, and named for the town where it was created. The script also has been known by different spellings of the town's name: Büthakukye or Beitha Kukju. Use of this alphabetic script arose between 1824 and 1845. The earliest use of Vithkuqi was in a nineteenth-century spelling book that was the basis for spelling books used in the regions of Bulgaria and Albania. A copy of the early Vithkuqi spelling book can be found in the Gennadius Library in Athens.

There are revivalist efforts in artistic and cultural uses of Vithkuqui, notably in the script's use in modern tattoos. Vithkuqi glyphs visually resemble cursive Armenian.

Structure. Vithkuqi is a left-to-right alphabetic script. There is no ligation, and non-productive diacritics are encoded atomically.

Casing. Casing is used in the Vithkuqi script.

Numerals and Punctuation. Vithkuqi uses European numbers and standard Latin punctuation.

8.13 Todhri

8.13.1 Todhri: U+105C0–U+105FF

The Todhri alphabet is a historical script used to write the Albanian language in the central Albanian region that is now designated Elbasan County. It was used in the eighteenth and nineteenth centuries and perhaps sporadically into the twentieth century. Theodor Haxhifilipi, a teacher at a Greek school whose nickname was Dhaskal Todhri, "teacher Todhri," is purported to be the creator. Extant material in the script is largely confined to copybooks of Biblical texts, although the script is said to have also been employed by merchants in their correspondence.

Structure. The Todhri script is an alphabet, written left to right, with no casing. The six consonants located from U+105EE..U+105F3 are used to represent the Greek characters γ *gamma*, ς *stigma*, ξ *ksi*, χ *chi*, ψ *psi*, and ω *omega*.

Vowels. The script has seven vowels, which are all atomically encoded. The two vowels U+105C9 i TODHRI LETTER EI and U+105E4 ȯ TODHRI LETTER U have canonical decompositions so that confusable sequences represented by a letter followed by a combining dot will be interpreted consistently. These relations are shown in *Figure 8-2*.

Figure 8-2. Todhri Vowel Decomposition

U+105C9 i TODHRI LETTER EI ≡ 105D2 ı + 0307 ȯ

U+105E4 ȯ TODHRI LETTER U ≡ 105DA o + 0307 ȯ

Combining diacritics only appear above vowels, in a practice which is reminiscent of Greek accents. U+0301 ◌́ COMBINING ACUTE ACCENT is used to represent the nearly vertical mark over a vowel. U+0304 ◌̄ COMBINING MACRON is employed for the horizontal diacritic over a single vowel, and U+035E ◌͞ COMBINING DOUBLE MACRON is used when the diacritic extends over two vowels. Smooth breathing is represented by U+0311 ◌̑ COMBINING INVERTED BREVE.

Punctuation. Manuscripts include comma and full stop, which should be represented by U+002C "," COMMA and U+002E "." FULL STOP.

8.14 Old Permic

8.14.1 Old Permic: U+10350–U+1037F

The Old Permic script was devised in the 14th century by the Russian missionary Stefan of Perm, and was used to write the Uralic languages Komi and Komi-Permyak. It was modeled on the Greek and Cyrillic alphabets, but many glyphs were taken from the "Tamga signs" used in indigenous Komi religious practices. Stefan translated Russian and Greek liturgical and biblical texts into Komi. There are a few surviving medieval documents in the script, chiefly in the form of icons, glosses, and inscriptions on monuments.

Old Permic continued to be used for Komi until the 17th century. In addition, the script was used cryptographically from the 15th century to write Russian, because it was unknown to most readers of Russian.

Structure. Old Permic is a simple, caseless, alphabetic script, read from left to right in horizontal lines running from top to bottom.

Combining Letters. A small number of letters, encoded from U+10376 to U+1037A, appear as superscript letters in abbreviations, in the same way that letters are used in Latin and Cyrillic.

Combining Marks. Old Permic employs a number of combining marks, as shown in *Table 8-2*. U+0483 COMBINING CYRILLIC TITLO indicates an abbreviation, typically with a specific set of holy words. The *combining grave accent* was sometimes used to mark consonant palatalization, and the *combining diaeresis* at times distinguishes [i] and [j]. However, the use of these combining marks is not always clear, and may have no phonetic value at all.

Table 8-2. Combining Marks Used in Old Permic

U+0300 COMBINING GRAVE ACCENT
U+0306 COMBINING BREVE
U+0307 COMBINING DOT ABOVE
U+0308 COMBINING DIAERESIS
U+0313 COMBINING COMMA ABOVE
U+0483 COMBINING CYRILLIC TITLO
U+20DB COMBINING THREE DOTS ABOVE

Numerals. Script-specific numerals are not known. Letters of the alphabet can be marked with U+0483 COMBINING CYRILLIC TITLO to indicate numeric use.

Punctuation. Old Permic does not have any script-specific punctuation, but uses middle dot, colon, and apostrophe. Spaces are used to separate words in manuscripts.

8.15 Ogham

8.15.1 Ogham: U+1680–U+169F

Ogham is an alphabetic script devised to write a very early form of Irish. Monumental Ogham inscriptions are found in Ireland, Wales, Scotland, England, and on the Isle of Man. Many of the Scottish inscriptions are undeciphered and may be in Pictish. It is probable that Ogham (Old Irish "Ogam") was widely written in wood in early times. The main flowering of "classical" Ogham, rendered in monumental stone, was in the fifth and sixth centuries CE. Such inscriptions were mainly employed as territorial markers and memorials; the more ancient examples are standing stones.

The script was originally written along the edges of stone where two faces meet; when written on paper, the central "stemlines" of the script can be said to represent the edge of the stone. Inscriptions written on stemlines cut into the face of the stone, instead of along its edge, are known as "scholastic" and are of a later date (post-seventh century). Notes were also commonly written in Ogham in manuscripts as recently as the sixteenth century.

Structure. The Ogham alphabet consists of 26 distinct characters (*feda*), the first 20 of which are considered to be primary and the last 6 (*forfeda*) supplementary. The four primary series are called *aicmí* (plural of *aicme*, meaning "family"). Each *aicme* was named after its first character, (*Aicme Beithe*, *Aicme Uatha*, meaning "the B Family," "the H Family," and so forth). The character names used in this standard reflect the spelling of the names in modern Irish Gaelic, except that the acute accent is stripped from *Úr*, *Éabhadh*, *Ór*, and *Ifín*, and the mutation of *nGéadal* is not reflected.

Rendering. Ogham text is read beginning from the bottom left side of a stone, continuing upward, across the top, and down the right side (in the case of long inscriptions). Monumental Ogham was incised chiefly in a bottom-to-top direction, though there are examples of left-to-right bilingual inscriptions in Irish and Latin. Manuscript Ogham accommodated the horizontal left-to-right direction of the Latin script, and the vowels were written as vertical strokes as opposed to the incised notches of the inscriptions. Ogham should therefore be rendered on computers from left to right or from bottom to top (never starting from top to bottom).

Forfeda (Supplementary Characters). In printed and in manuscript Ogham, the fonts are conventionally designed with a central stemline, but this convention is not necessary. In implementations without the stemline, the character U+1680 OGHAM SPACE MARK should be given its conventional width and simply left blank like U+0020 SPACE. U+169B OGHAM FEATHER MARK and U+169C OGHAM REVERSED FEATHER MARK are used at the beginning and the end of Ogham text, particularly in

manuscript Ogham. In some cases, only the *Ogham feather mark* is used, which can indicate the direction of the text.

The word *latheirt* ᚛ᚂᚐᚈᚆᚓᚔᚏᚈ᚜ shows the use of the feather marks. This word was written in the margin of a ninth-century Latin grammar and means "massive hangover," which may be the scribe's apology for any errors in his text.

8.16 Shavian

8.16.1 Shavian: U+10450–U+1047F

The playwright George Bernard Shaw (1856–1950) was an outspoken critic of the idiosyncrasies of English orthography. In his will, he directed that Britain's Public Trustee seek out and publish an alphabet of no fewer than 40 letters to provide for the phonetic spelling of English. The alphabet finally selected was designed by Kingsley Read and is variously known as Shavian, Shaw's alphabet, and the Proposed British Alphabet. Also in accordance with Shaw's will, an edition of his play, *Androcles and the Lion*, was published and distributed to libraries, containing the text both in the standard Latin alphabet and in Shavian.

As with other attempts at spelling reform in English, the alphabet has met with little success. Nonetheless, it has its advocates and users. The normative version of Shavian is taken to be the version in *Androcles and the Lion*.

Structure. The alphabet consists of 48 letters and 1 punctuation mark. The letters have no case. The digits and other punctuation marks are the same as for the Latin script. The one additional punctuation mark is a "name mark," used to indicate proper nouns. U+00B7 MIDDLE DOT should be used to represent the "name mark." The letter names are intended to be indicative of their sounds; thus the sound /p/ is represented by U+10450 𐑐 SHAVIAN LETTER PEEP.

The first 40 letters are divided into four groups of 10. The first 10 and second 10 are 180-degree rotations of one another; the letters of the third and fourth groups often show a similar relationship of shape.

The first 10 letters are tall letters, which ascend above the x-height and generally represent unvoiced consonants. The next 10 letters are "deep" letters, which descend below the baseline and generally represent voiced consonants. The next 20 are the vowels and liquids. Again, each of these letters usually has a close phonetic relationship to the letter in its matching set of 10.

The remaining 8 letters are technically ligatures, the first 6 involving vowels plus /r/. Because ligation is not optional, these 8 letters are included in the encoding.

Collation. The problem of collation is not addressed by the alphabet's designers.

Chapter 9

Middle East-I

Modern and Liturgical Scripts

Most scripts in this chapter have a common origin in the ancient Phoenician alphabet.

The Hebrew script is used in Israel and for languages of the Diaspora. The Arabic script is used to write many languages throughout the Middle East, North Africa, and certain parts of Asia. The Syriac script is used to write a number of Middle Eastern languages. These three also function as major liturgical scripts, used worldwide by various religious groups.

The Samaritan script is used in small communities in Israel and the Palestinian Territories to write the Samaritan Hebrew and Samaritan Aramaic languages. The Mandaic script was used in southern Mesopotamia in classical times for liturgical texts by adherents of the Mandaean gnostic religion. The Classical Mandaic and Neo-Mandaic languages are still in limited current use in modern Iran and Iraq and in the Mandaean diaspora.

Unlike most of the other scripts discussed in this chapter, the Yezidi script is an alphabet. The script was used to write two religious texts which may date to the 12th or 13th centuries. The script was recently revived and is used by clergy in the Yezidi temple in Tbilisi.

The Middle Eastern scripts are mostly abjads, with small character sets. Words are demarcated by spaces. These scripts include a number of distinctive punctuation marks. In addition, the Arabic script includes traditional forms for digits, called "Arabic-Indic digits" in the Unicode Standard.

Text in these scripts is written from right to left. Implementations of these scripts must conform to the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm"). For more information about writing direction, see *Section 2.10, Writing Direction*. There are also special security considerations that apply to bidirectional scripts, especially with regard to their use in identifiers. For more information about these issues, see Unicode Technical Report #36, "Unicode Security Considerations."

Arabic, Syriac and Mandaic are cursive scripts even when typeset, unlike Hebrew and Samaritan, where letters are unconnected. Most letters in Arabic, Syriac and Mandaic assume different forms depending on their position in a word. Shaping rules for the rendering of text are specified in *Section 9.2, Arabic*, *Section 9.3, Syriac* and *Section 9.5, Mandaic*. Shaping rules are not required for Hebrew because only five letters have position-dependent final forms, and these forms are separately encoded.

Historically, Middle Eastern scripts did not write short vowels. Nowadays, short vowels are represented by marks positioned above or below a consonantal letter. Vowels and other pronunciation ("vocalization") marks are encoded as combining characters, so support for vocalized text necessitates use of composed character sequences. Yiddish and Syriac are normally written with vocalization; Hebrew, Samaritan, and Arabic are usually written unvocalized.

9.1 Hebrew

9.1.1 Hebrew: U+0590–U+05FF

The Hebrew script is used for writing the Hebrew language as well as Yiddish, Judezmo (Ladino), and a number of other languages. Vowels and various other marks are written as *points,* which are applied to consonantal base letters; these marks are usually omitted in Hebrew, except for liturgical texts and other special applications. Five Hebrew letters assume a different graphic form when they occur last in a word.

Directionality. The Hebrew script is written from right to left. Conformant implementations of Hebrew script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Cursive. The Unicode Standard uses the term *cursive* to refer to writing where the letters of a word are connected. A handwritten form of Hebrew is known as cursive, but its rounded letters are generally unconnected, so the Unicode definition does not apply. Fonts based on cursive Hebrew exist. They are used not only to show examples of Hebrew handwriting, but also for display purposes.

Standards. ISO/IEC 8859-8—Part 8. *Latin/Hebrew Alphabet*. The Unicode Standard encodes the Hebrew alphabetic characters in the same relative positions as in ISO/IEC 8859-8; however, there are no points or Hebrew punctuation characters in that ISO standard.

Vowels and Other Pronunciation Marks. These combining marks, generically called *points* in the context of Hebrew, indicate vowels or other modifications of consonantal letters. General rules for applying combining marks are given in *Section 2.11, Combining Characters*, and *Section 3.6, Combination*. Additional Hebrew-specific behavior is described below.

Hebrew points can be separated into four classes: *dagesh*, *shin dot* and *sin dot*, vowels, and other pronunciation marks.

Dagesh, U+05BC HEBREW POINT DAGESH OR MAPIQ, has the form of a dot that appears inside the letter that it affects. It is not a vowel but rather a diacritic that affects the pronunciation of a consonant. The same base consonant can also have a vowel and/or other diacritics. *Dagesh* is the only element that goes inside a letter.

The dotted Hebrew consonant *shin* is explicitly encoded as the sequence U+05E9 HEBREW LETTER SHIN followed by U+05C1 HEBREW POINT SHIN DOT. The *shin dot* is positioned on the upper-right side of the undotted base letter. Similarly, the dotted consonant *sin* is explicitly encoded as the sequence U+05E9 HEBREW LETTER SHIN followed by U+05C2 HEBREW POINT SIN DOT. The *sin dot* is positioned on the upper-

left side of the base letter. The two dots are mutually exclusive. The base letter *shin* can also have a *dagesh*, a vowel, and other diacritics. The two dots are not used with any other base character.

Vowels all appear below the base character that they affect, except for *holam*, U+05B9 HEBREW POINT HOLAM, which appears above left. The following points represent vowels: U+05B0..U+05BB, and U+05C7.

The remaining three points are *pronunciation marks*: U+05BD HEBREW POINT METEG, U+05BF HEBREW POINT RAFE, and U+FB1E HEBREW POINT JUDEO-SPANISH VARIKA. *Meteg*, also known as *siluq*, goes below the base character; *rafe* and *varika* go above it. The varika, used in Judezmo, is a glyphic variant of *rafe*.

Shin and Sin. Separate characters for the dotted letters *shin* and *sin* are not included in this block. When it is necessary to distinguish between the two forms, they should be encoded as U+05E9 HEBREW LETTER SHIN followed by the appropriate dot, either U+05C1 HEBREW POINT SHIN DOT or U+05C2 HEBREW POINT SIN DOT. (See preceding discussion.) This practice is consistent with Israeli standard encoding.

Final (Contextual Variant) Letterforms. Variant forms of five Hebrew letters are encoded as separate characters in this block, as in Hebrew standards including ISO/IEC 8859-8. These variant forms are generally used in place of the nominal letter-forms at the end of words. Certain words, however, are spelled with nominal rather than final forms, particularly names and foreign borrowings in Hebrew and some words in Yiddish. Because final form usage is a matter of spelling convention, software should not automatically substitute final forms for nominal forms at the end of words. The positional variants should be coded directly and rendered one-to-one via their own glyphs—that is, without contextual analysis.

Yiddish Digraphs. The digraphs are considered to be independent characters in Yiddish. The Unicode Standard has included them as separate characters so as to distinguish certain letter combinations in Yiddish text—for example, to distinguish the digraph *double vav* from an occurrence of a consonantal *vav* followed by a vocalic *vav*. The use of digraphs is consistent with standard Yiddish orthography. Other letters of the Yiddish alphabet, such as *pasekh alef,* can be composed from other characters, although alphabetic presentation forms are also encoded.

Punctuation. Most punctuation marks used with the Hebrew script are not given independent codes (that is, they are unified with Latin punctuation) except for the few cases where the mark has a unique form in Hebrew—namely, U+05BE HEBREW PUNCTUATION MAQAF, U+05C0 HEBREW PUNCTUATION PASEQ (also known as *legarmeh*), U+05C3 HEBREW PUNCTUATION SOF PASUQ, U+05F3 HEBREW PUNCTUATION GERESH, and U+05F4 HEBREW PUNCTUATION GERSHAYIM. For paired punctuation such as parentheses, the glyphs chosen to represent U+0028 LEFT PARENTHESIS and U+0029 RIGHT PARENTHESIS will depend on the direction of the

rendered text. See *Section 4.7, Bidi Mirrored*, for more information. For additional punctuation to be used with the Hebrew script, see *Section 6.2, General Punctuation*.

Cantillation Marks. Cantillation marks are used in publishing liturgical texts, including the Bible. There are various historical schools of cantillation marking; the set of marks included in the Unicode Standard follows the Israeli standard SI 1311.2.

Positioning. Marks may combine with vowels and other points, and complex typographic rules dictate how to position these combinations.

The vertical placement (meaning above, below, or inside) of points and marks is very well defined. The horizontal placement (meaning left, right, or center) of points is also very well defined. The horizontal placement of marks, by contrast, is not well defined, and convention allows for the different placement of marks relative to their base character.

When points and marks are located below the same base letter, the point always comes first (on the right) and the mark after it (on the left), except for the marks *yetiv*, U+059A HEBREW ACCENT YETIV, and *dehi*, U+05AD HEBREW ACCENT DEHI. These two marks come first (on the right) and are followed (on the left) by the point.

These rules are followed when points and marks are located above the same base letter:

- If the point is *holam*, all cantillation marks precede it (on the right) except *pashta*, U+0599 HEBREW ACCENT PASHTA.

- *Pashta* always follows (goes to the left of) points.

- *Holam* on a *sin* consonant (*shin* base + *sin dot*) follows (goes to the left of) the *sin dot*. However, the two combining marks are sometimes rendered as a single assimilated dot.

- *Shin dot* and *sin dot* are generally represented closer vertically to the base letter than other points and marks that go above it.

Meteg. *Meteg*, U+05BD HEBREW POINT METEG, frequently co-occurs with vowel points below the consonant. Typically, *meteg* is placed to the left of the vowel, although in some manuscripts and printed texts it is positioned to the right of the vowel. The difference in positioning is not known to have any semantic significance; nevertheless, some authors wish to retain the positioning found in source documents.

The alternate *vowel-meteg* ordering can be represented in terms of alternate ordering of characters in encoded representation. However, because of the fixed-position canonical combining classes to which *meteg* and vowel points are assigned, differences in ordering of such characters are not preserved under normalization. The *combining grapheme joiner* can be used within a *vowel-meteg* sequence to preserve an

ordering distinction under normalization. For more information, see the description of U+034F COMBINING GRAPHEME JOINER in *Section 23.2, Layout Controls*.

For example, to display *meteg* to the left of (after, for a right-to-left script) the vowel point *sheva*, U+05B0 HEBREW POINT SHEVA, the sequence of *meteg* following *sheva* can be used:

> <sheva, meteg>

Because these marks are canonically ordered, this sequence is preserved under normalization. Then, to display *meteg* to the right of the *sheva*, the sequence with *meteg* preceding *sheva* with an intervening CGJ can be used:

> <meteg, CGJ, sheva>

A further complication arises for combinations of *meteg* with *hataf* vowels: U+05B1 HEBREW POINT HATAF SEGOL, U+05B2 HEBREW POINT HATAF PATAH, and U+05B3 HEBREW POINT HATAF QAMATS. These vowel points have two side-by-side components. *Meteg* can be placed to the left or the right of a *hataf* vowel, but it also is often placed between the two components of the *hataf* vowel. A three-way positioning distinction is needed for such cases.

The *combining grapheme joiner* can be used to preserve an ordering that places *meteg* to the right of a *hataf* vowel, as described for combinations of *meteg* with non-*hataf* vowels, such as *sheva*.

Placement of *meteg* between the components of a *hataf* vowel can be conceptualized as a ligature of the *hataf* vowel and a nominally positioned *meteg*. With this in mind, the ligation-control functionality of U+200D ZERO WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER can be used as a mechanism to control the visual distinction between a nominally positioned *meteg* to the left of a *hataf* vowel versus the medially positioned *meteg* within the *hataf* vowel. That is, *zero width joiner* can be used to request explicitly a medially positioned *meteg*, and *zero width non-joiner* can be used to request explicitly a left-positioned *meteg*. Just as different font implementations may or may not display an "fi" ligature by default, different font implementations may or may not display *meteg* in a medial position when combined with *hataf* vowels by default. As a result, authors who want to ensure left-position versus medial-position display of *meteg* with *hataf* vowels across all font implementations may use joiner characters to distinguish these cases.

Thus the following encoded representations can be used for different positioning of *meteg* with a *hataf* vowel, such as *hataf patah*:

> left-positioned *meteg*: <hataf patah, ZWNJ, meteg>

> medially positioned *meteg*: <hataf patah, ZWJ, meteg>

right-positioned *meteg*: <meteg, CGJ, hataf patah>

In no case is use of ZWNJ, ZWJ, or CGJ *required* for representation of *meteg*. These recommendations are simply provided for interoperability in those instances where authors wish to preserve specific positional information regarding the layout of a *meteg* in text.

Atnah Hafukh and Qamats Qatan. In some older versions of Biblical text, a distinction is made between the accents U+05A2 HEBREW ACCENT ATNAH HAFUKH and U+05AA HEBREW ACCENT YERAH BEN YOMO. Many editions from the last few centuries do not retain this distinction, using only *yerah ben yomo*, but some users in recent decades have begun to reintroduce this distinction. Similarly, a number of publishers of Biblical or other religious texts have introduced a typographic distinction for the vowel point *qamats* corresponding to two different readings. The original letterform used for one reading is referred to as *qamats* or *qamats gadol*; the new letterform for the other reading is *qamats qatan*. Not all users of Biblical Hebrew use *atnah hafukh* and *qamats qatan*. If the distinction between accents *atnah hafukh* and *yerah ben yomo* is not made, then only U+05AA HEBREW ACCENT YERAH BEN YOMO is used. If the distinction between vowels *qamats gadol* and *qamats qatan* is not made, then only U+05B8 HEBREW POINT QAMATS is used. Implementations that support Hebrew accents and vowel points may not necessarily support the special-usage characters U+05A2 HEBREW ACCENT ATNAH HAFUKH and U+05C7 HEBREW POINT QAMATS QATAN.

Holam Male and Holam Haser. The vowel point *holam* represents the vowel phoneme /o/. The consonant letter *vav* represents the consonant phoneme /w/, but in some words is used to represent a vowel, /o/. When the point *holam* is used on *vav*, the combination usually represents the vowel /o/, but in a very small number of cases represents the consonant-vowel combination /wo/. A typographic distinction is made between these two in many versions of Biblical text. In most cases, in which *vav* + *holam* together represents the vowel /o/, the point *holam* is centered above the *vav* and referred to as *holam male*. In the less frequent cases, in which the *vav* represents the consonant /w/, some versions show the point *holam* positioned above left. This is referred to as *holam haser*. The character U+05BA HEBREW POINT HOLAM HASER FOR VAV is intended for use as *holam haser* only in those cases where a distinction is needed. When the distinction is made, the character U+05B9 HEBREW POINT HOLAM is used to represent the point *holam male on vav*. U+05BA HEBREW POINT HOLAM HASER FOR VAV is intended for use only on *vav*; results of combining this character with other base characters are not defined. Not all users distinguish between the two forms of *holam*, and not all implementations can be assumed to support U+05BA HEBREW POINT HOLAM HASER FOR VAV.

Puncta Extraordinaria. In the Hebrew Bible, dots are written in various places above or below the base letters that are distinct from the vowel points and accents. These dots are referred to by scholars as *puncta extraordinaria*, and there are two

kinds. The *upper punctum*, the more common of the two, has been encoded since Unicode 2.0 as U+05C4 HEBREW MARK UPPER DOT. The *lower punctum* is used in only one verse of the Bible, Psalm 27:13, and is encoded as U+05C5 HEBREW MARK LOWER DOT. The *puncta* generally differ in appearance from dots that occur above letters used to represent numbers; the number dots should be represented using U+0307 COMBINING DOT ABOVE and U+0308 COMBINING DIAERESIS.

Nun Hafukha. The *nun hafukha* is a special symbol that appears to have been used for scribal annotations, although its exact functions are uncertain. It is used a total of nine times in the Hebrew Bible, although not all versions include it, and there are variations in the exact locations in which it is used. There is also variation in the glyph used: it often has the appearance of a rotated or reversed *nun* and is very often called *inverted nun*; it may also appear similar to a *half tet* or have some other form. In pointed texts, the *nun hafukha* carries a dot above it. This dot should be represented using U+0307 COMBINING DOT ABOVE.

Currency Symbol. The NEW SHEQEL SIGN (U+20AA) is encoded in the currency block.

9.1.2 Alphabetic Presentation Forms: U+FB00–U+FB4F

The Hebrew characters in this block are chiefly of two types: variants of letters and marks encoded in the main Hebrew block, and precomposed combinations of a Hebrew letter or digraph with one or more vowels or pronunciation marks. This block contains all of the vocalized letters of the Yiddish alphabet. The *alef lamed* ligature and a Hebrew variant of the plus sign are included as well. The Hebrew plus sign variant, U+FB29 HEBREW LETTER ALTERNATIVE PLUS SIGN, is used more often in handwriting than in print, but it does occur in school textbooks. It is used by those who wish to avoid cross symbols, which can have religious and historical connotations.

U+FB20 HEBREW LETTER ALTERNATIVE AYIN is an alternative form of *ayin* that may replace the basic form U+05E2 HEBREW LETTER AYIN when there is a diacritical mark below it. The basic form of *ayin* is often designed with a descender, which can interfere with a mark below the letter. U+FB20 is encoded for compatibility with implementations that substitute the alternative form in the character data, as opposed to using a substitute glyph at rendering time.

Use of Wide Letters. Wide letterforms are used in handwriting and in print to achieve even margins. The wide-form letters in the Unicode Standard are those that are most commonly "stretched" in justification. If Hebrew text is to be rendered with even margins, justification should be left to the text-formatting software.

These alphabetic presentation forms are included for compatibility purposes. For the preferred encoding, see the Hebrew presentation forms, U+FB1D..U+FB4F.

For letterlike symbols, see U+2135..U+2138.

9.2 Arabic

9.2.1 Arabic: U+0600–U+06FF

The Arabic script is used for writing the Arabic language and has been extended to represent a number of other languages, such as Persian, Urdu, Pashto, Sindhi, and Uyghur, as well as many African languages. Urdu is often written with the ornate Nastaliq script variety. Some languages, such as Indonesian/Malay, Turkish, and Ingush, formerly used the Arabic script but now employ the Latin or Cyrillic scripts. Other languages, such as Kurdish, Azerbaijani, Kazakh, and Uzbek have competing Arabic and Latin or Cyrillic orthographies in different countries.

The Arabic script is cursive, even in its printed form (see *Figure 9-1*). As a result, the same letter may be written in different forms depending on how it joins with its neighbors. Vowels and various other marks may be written as combining marks called *tashkil*, which are applied to consonantal base letters. In normal writing, however, these marks are omitted.

Figure 9-1. Directionality and Cursive Connection

Memory representation:	ههـ ه
After reordering:	ه ـههه
After joining:	ههه ه

Directionality. The Arabic script is written from right to left. Conformant implementations of Arabic script must use the Unicode Bidirectional Algorithm to reorder the memory representation for display (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Standards. ISO/IEC 8859-6—Part 6. *Latin/Arabic Alphabet*. The Unicode Standard encodes the basic Arabic characters in the same relative positions as in ISO/IEC 8859-6. ISO/IEC 8859-6, in turn, is based on ECMA-114, which was based on ASMO 449.

Encoding Principles. The basic set of Arabic letters is well defined. Each letter receives only one Unicode character value in the basic Arabic block, no matter how many different contextual appearances it may exhibit in text. Each Arabic letter in the Unicode Standard may be said to represent the inherent semantic identity of the letter. A word is spelled as a sequence of these letters. The representative glyph shown in the Unicode character chart for an Arabic letter is usually the form of the letter when standing by itself. It is simply used to distinguish and identify the character in the code charts and does not restrict the glyphs used to represent it. See "Arabic Cursive Joining," "Arabic Ligatures," and "Arabic Joining Groups" in the

following text for an extensive discussion of how cursive joining and positional variants of Arabic letters are handled by the Unicode Standard.

The following principles guide the encoding of the various types of marks which are applied to the basic Arabic letter skeletons:

1. **Ijam**: Diacritical marks applied to basic letter forms to derive new (usually consonant) letters for extended Arabic alphabets are not separately encoded as combining marks. Instead, each letter plus *ijam* combination is encoded as a separate, atomic character. These letter plus *ijam* characters are never given decompositions in the standard. *Ijam* generally take the form of one-, two-, three- or four-dot markings above or below the basic letter skeleton, although other diacritic forms occur in extensions of the Arabic script in Central and South Asia and in Africa. In discussions of Arabic in Unicode, *ijam* are often also referred to as *nukta*, because of their functional similarity to the nukta diacritical marks which occur in many Indic scripts.

2. **Tashkil**: Marks functioning to indicate vocalization of text, as well as other types of phonetic guides to correct pronunciation, are separately encoded as combining marks. These include several subtypes: *harakat* (short vowel marks), *tanwin* (postnasalized or long vowel marks), *shaddah* (consonant gemination mark), and *sukun* (to mark lack of a following vowel). A basic Arabic letter plus any of these types of marks is never encoded as a separate, precomposed character, but must always be represented as a sequence of letter plus combining mark. Additional marks invented to indicate non-Arabic vowels, used in extensions of the Arabic script, are also encoded as separate combining marks.

3. **Maddah**: The *maddah* is a particular case of a *harakat* mark which has exceptional treatment in the standard. In most modern languages using the Arabic script, it occurs only above *alef*, and in that combination represents the sound /ʔaa/. In Quranic Arabic, *maddah* occurs above *waw* or *yeh* to note vowel elongation. For this reason, and the shared use of *maddah* between Arabic and Syriac scripts, the precomposed combination U+0622 ARABIC LETTER ALEF WITH MADDA ABOVE is encoded, however the combining mark U+0653 ARABIC MADDAH ABOVE is also encoded. U+0622 is given a canonical decomposition to the sequence of *alef* followed by the *combining maddah*. Some historical non-Arabic orthographies have also used *maddah* as an *ijam*. U+0653 should be used to represent those texts.

4. **Hamza**: The *hamza* may occur above or below other letters. Its treatment in the Unicode Standard is also exceptional and rather complex. The general principle is that when such a *hamza* is used to indicate an actual glottal stop, the /je/ sound used in Persian and Urdu for *ezafe*, or the short vowels /ə/ and /ɨ/ in Kashmiri, it should be represented with a separate combining mark,

either U+0654 ARABIC HAMZA ABOVE or U+0655 ARABIC HAMZA BELOW. However, when the *hamza* mark is used as a diacritic to derive a separate letter as an extension of the Arabic script, then the basic letter skeleton plus the *hamza* mark is represented by a single, precomposed character. See "Combining Hamza Above" later in this section for discussion of the complications for particular characters.

5. **Annotation Marks**: Quranic annotation marks are always encoded as separate combining marks.

Punctuation. Most punctuation marks used with the Arabic script are not given independent codes (that is, they are unified with Latin punctuation), except for the few cases where the mark has a significantly different appearance in Arabic—namely, U+060C ARABIC COMMA, U+061B ARABIC SEMICOLON, U+061E ARABIC TRIPLE DOT PUNCTUATION MARK, U+061F ARABIC QUESTION MARK, and U+066A ARABIC PERCENT SIGN.

Persian and some other languages use rounded forms of U+00AB LEFT-POINTING DOUBLE ANGLE QUOTATION MARK and U+00BB RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK.

Sindhi uses U+2E41 REVERSED COMMA and U+204F REVERSED SEMICOLON. Some fonts have used glyph variants of U+060C ARABIC COMMA and U+061B ARABIC SEMICOLON, although this is not recommended.

For paired punctuation such as parentheses, the glyphs chosen to display for example, U+0028 LEFT PARENTHESIS and U+0029 RIGHT PARENTHESIS, will depend on the direction of the rendered text. See "Paired Punctuation" in *Section 6.2, General Punctuation*, for more discussion.

The Non-joiner and the Joiner. The Unicode Standard provides two user-selectable formatting codes: U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER. The use of a joiner adjacent to a suitable letter permits that letter to form a cursive connection without a visible neighbor. This provides a simple way to encode some special cases, such as exhibiting a connecting form in isolation, as shown in *Figure 9-2*.

Figure 9-2. Using a Joiner

Memory representation:	ههه ‍ههـ ه
After reordering:	ه ههـ ‍ ههه
After joining:	ﻪ ‍ ﻬﻪ ه

These connecting forms commonly occur in some abbreviations such as the marker for *hijri* dates, which consists of an initial form of *heh*: ﻫ .

The use of a non-joiner between two letters prevents those letters from forming a cursive connection with each other when rendered, as shown in *Figure 9-3*.

Figure 9-3. Using a Non-joiner

Memory representation: ھ ـﻬﻬ ‌﮿ ھ

After reordering: ھ ‌﮿ ﻬﻬ ھ

After joining: ھ ‌ ھھھ ھ

Examples requiring the use of a non-joiner include the Persian plural suffix, some Persian proper names, and Ottoman Turkish vowels. This use of non-joiners is important for representation of text in such languages, and ignoring or removing them will result in text with a different meaning, or in meaningless text.

Joiners and non-joiners may also occur in combinations. The effects of such combinations are shown in *Figure 9-4*. For further discussion of joiners and non-joiners, see *Section 23.2, Layout Controls*.

Figure 9-4. Combinations of Joiners and Non-joiners

Memory representation: ھ ‌﮿ ‌﮿ ﻬﻬ ﮿ ھ

After reordering: ھ ﮿ ﻬﻬ ‌﮿ ‌﮿ ھ

After joining: ھ ﮿ ھھھ

Tashkil Nonspacing Marks. *Tashkil* are marks that indicate vowels or other modifications of consonant letters. In English, these marks are often referred to as "points." They may also be called *harakat*, although technically, *harakat* refers to the subset of *tashkil* which denote short vowels. The code charts depict these *tashkil* in relation to a dotted circle, indicating that this character is intended to be applied via some process *to the character that precedes it* in the text stream (that is, the base character). General rules for applying nonspacing marks are given in *Section 7.9, Combining Marks*. The few marks that are placed after (to the left of) the base character are treated as ordinary spacing characters in the Unicode Standard. For more information about the canonical ordering of nonspacing marks, see *Section 2.11, Combining Characters,* and *Section 3.11, Normalization Forms*.

Use of the Arabic Mark Transient Reordering Algorithm (AMTRA) during text display is recommended to correctly and consistently render Arabic combining mark sequences. This algorithm provides results that match user expectations and assures that canonically equivalent sequences are rendered identically, independent of the order of the combining marks in the text stream. For more information, see Unicode Technical Report #53, "Unicode Arabic Mark Rendering."

The placement and rendering of vowel and other marks in Arabic strongly depends on the typographical environment or even the typographical style. For example, in the Unicode code charts, the default position of U+0651 � ARABIC SHADDA is with the glyph placed above the base character, whereas for U+0650 ٍ ARABIC KASRA the glyph is placed below the base character, as shown in the first example in *Figure 9-5*. However, computer fonts often follow an approach that originated in metal typesetting and combine the *kasra* with *shadda* above the text, as shown in the second example in *Figure 9-5*. U+064D ٍ ARABIC KASRATAN also follows this behavior.

Figure 9-5. Placement of Harakat

The shapes of the various *tashkil* marks may also depend on the style of writing. For example, *dammatan* can be written in at least four different styles, as shown in *Figure 9-6*.

Figure 9-6. Dammatan Styles
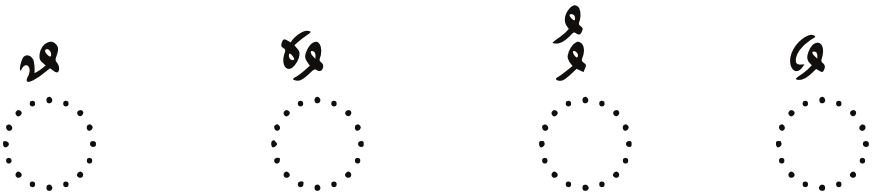
U+064C ARABIC DAMMATAN can be rendered in any of those four shapes. U+08F1 ARABIC OPEN DAMMATAN is an alternative *dammatan* character for use in Quran orthographies which have two distinct forms of *dammatan* that convey a semantic difference.

Arabic-Indic Digits. The names for the forms of decimal digits vary widely across different languages. The decimal numbering system originated in India (Devanagari ०१२३...) and was subsequently adopted in the Arabic world with a different appearance (Arabic ٠١٢٣...). The Europeans adopted decimal numbers from the Arabic world, although once again the forms of the digits changed greatly (European 0123...). The European forms were later adopted widely around the world and are used even in many Arabic-speaking countries in North Africa. In each case, the interpretation of decimal numbers remained the same. However, the forms of the digits changed to such a degree that they are no longer recognizably the same characters. Because of the origin of these characters, the European decimal numbers are widely

known as "Arabic numerals" or "Hindi-Arabic numerals," whereas the decimal numbers in use in the Arabic world are widely known there as "Hindi numbers."

The Unicode Standard includes *Indic* digits (including forms used with different Indic scripts), *Arabic* digits (with forms used in most of the Arabic world), and *European* digits (now used internationally). Because of this decision, the traditional names could not be retained without confusion. In addition, there are two main variants of the Arabic digits: those used in Afghanistan, India, Iran, and Pakistan (here called *Eastern Arabic-Indic*) and those used in other parts of the Arabic world. In summary, the Unicode Standard uses the names shown in <u>*Table 9-1*</u>. A different set of number forms, called Rumi, was used in historical materials from Egypt to Spain, and is discussed in the subsection on "Rumi Numeral Symbols" in <u>*Section 22.3, Numerals*</u>.

Table 9-1. Arabic Digit Names

Name	Code Points	Forms
European	U+0030..U+0039	0123456789
Arabic-Indic	U+0660..U+0669	٠١٢٣٤٥٦٧٨٩
Eastern Arabic-Indic	U+06F0..U+06F9	۰۱۲۳۴۵۶۷۸۹
Indic (Devanagari)	U+0966..U+096F	०१२३४५६७८९

There are distinct glyph forms for Eastern Arabic-Indic digits for the digits four, five, six, and seven. Furthermore, for four, six, and seven, there is substantial variation between locales using the Eastern Arabic-Indic digits. <u>*Table 9-2*</u> illustrates this variation with some example glyphs for digits in languages of Afghanistan, India, Iran, and Pakistan. While some usage of the Persian glyph for U+06F7 EXTENDED ARABIC-INDIC DIGIT SEVEN can be documented for Sindhi, the form shown in <u>*Table 9-2*</u> is predominant.

Table 9-2. Glyph Variation in Eastern Arabic-Indic Digits

Code Point	Digit	Persian	Sindhi	Urdu and Kashmiri
U+06F4	4	۴	۴	۲
U+06F5	5	۵	۵	۵
U+06F6	6	۶	٦	٦
U+06F7	7	۷	٧	٧

The Unicode Standard provides a single, complete sequence of digits for Persian, Sindhi, and Urdu to account for the differences in appearance and directional treatment when rendering them. The Kashmiri digits have the same appearance as those for Urdu. (For a complete discussion of directional formatting of numbers in the Unicode Standard, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm.")

Extended Arabic Letters. Arabic script is used to write major languages, such as Persian and Urdu, but it has also been used to transcribe some lesser-used languages, such as Baluchi and Lahnda, which have little tradition of printed typography. As a result, the Unicode Standard encodes multiple forms of some Extended Arabic letters because the character forms and usages are not well documented for a number of languages. For additional extended Arabic letters, see the Arabic Supplement block, U+0750..U+077F, the Arabic Extended-A block, U+08A0..U+08FF, and the Arabic Extended-B block, U+0870..U+089F.

Quranic Annotation Signs. These characters are used in the Quran to mark pronunciation and other annotation of the text. Several additional Quranic annotation signs are encoded in the Arabic Extended-A block, U+08A0..U+08FF, and the Arabic Extended-B block, U+0870..U+089F. The alternate forms of some of the marks are not merely decorative; they are used to show variations in pronunciation (U+08F0 ARABIC OPEN FATHATAN), or can indicate various pause points (U+0615 ARABIC SMALL HIGH TAH), extended pauses, mandatory pauses, or places where a breath should not occur. They are required to guide the reader in reciting the text correctly. Some marks, such as U+08D4 ARABIC SMALL HIGH WORD AR-RUB, may be positioned above *end of ayah*.

Additional Vowel Marks. When the Arabic script is adopted as the writing system for a language other than Arabic, it is often necessary to represent vowel sounds or distinctions not made in Arabic. In some cases, conventions such as the addition of small dots above and/or below the standard Arabic *fatha*, *damma*, and *kasra* signs have been used.

Classical Arabic has only three canonical vowels (/a/, /i/, /u/), whereas languages such as Urdu and Persian include other contrasting vowels such as /o/ and /e/. For this reason, it is imperative that speakers of these languages be able to show the difference between /e/ and /i/ (U+0656 ARABIC SUBSCRIPT ALEF), and between /o/ and /u/ (U+0657 ARABIC INVERTED DAMMA). At the same time, the use of these two diacritics in Arabic is redundant, merely emphasizing that the underlying vowel is long.

U+065F ARABIC WAVY HAMZA BELOW is an additional vowel mark used in Kashmiri. It can appear in combination with many characters. The particular combination of an *alef* with this vowel mark should be written with the sequence <U+0627 ARABIC LETTER ALEF, U+065F ARABIC WAVY HAMZA BELOW>, rather than with the character U+0673 ARABIC LETTER ALEF WITH WAVY HAMZA BELOW, which has been deprecated and which is not canonically equivalent. However, implementations should be aware that there may be existing legacy Kashmiri data in which U+0673 occurs.

Honorifics. Marks known as honorifics represent phrases expressing the status of a person and are in widespread use in the Arabic-script world. Most have a specifically

religious meaning. In effect, these marks are combining characters at the word level, rather than being associated with a single base character. The normal practice is that such marks be used at the end of words. In manuscripts, depending on the letter shapes present in the name and the calligraphic style in use, the honorific mark may appear over a letter in the middle of the word. If an exact representation of a manuscript is desired, the honorific mark may be represented as following that letter. The normalization algorithm does not move such word-level combining characters to the end of the word.

Spacing honorifics are also in wide use both in the Arabic script and among Muslim communities writing in other scripts. See "Word Ligatures" under Arabic Presentation Forms-A later in this section for more information.

Arabic Mathematical Symbols. A few Arabic mathematical symbols are encoded in this block. The Arabic mathematical radix signs, U+0606 ARABIC-INDIC CUBE ROOT and U+0607 ARABIC-INDIC FOURTH ROOT, differ from simple mirrored versions of U+221B CUBE ROOT and U+221C FOURTH ROOT, in that the digit portions of the symbols are written with Arabic-Indic digits and are not mirrored. U+0608 ARABIC RAY is a letterlike symbol used in Arabic mathematics.

Date Separator. U+060D ARABIC DATE SEPARATOR is used in Pakistan and India between the numeric date and the month name when writing out a date. This sign is distinct from U+002F SOLIDUS, which is used, for example, as a separator in currency amounts.

Full Stop. U+061E ARABIC TRIPLE DOT PUNCTUATION MARK is encoded for traditional orthographic practice using the Arabic script to write African languages such as Hausa, Wolof, Fulani, and Mandinka. These languages use ARABIC TRIPLE DOT PUNCTUATION MARK as a full stop.

Currency Symbols. U+060B AFGHANI SIGN is a currency symbol used in Afghanistan. The symbol is derived from an abbreviation of the name of the currency, which has become a symbol in its own right. U+FDFC RIAL SIGN is a currency symbol used in Iran. Unlike the AFGHANI SIGN, U+FDFC RIAL SIGN is considered a compatibility character, encoded for compatibility with Iranian standards. Ordinarily in Persian "rial" is simply spelled out as the sequence of letters, <0631, 06CC, 0627, 0644>.

Signs Spanning Numbers. Several other special signs are written in association with numbers in the Arabic script. All of these signs can span multiple-digit numbers, rather than just a single digit. They are not formally considered *combining marks* in the sense used by the Unicode Standard, although they clearly interact graphically with their associated sequence of digits. In the text representation they *precede* the sequence of digits that they span, rather than follow a base character, as would be the case for a combining mark. Their General_Category value is Cf (format character).

Unlike most other format characters, however, they should be rendered with a visible glyph, even in circumstances where no suitable digit or sequence of digits follows them in logical order. The characters have the Bidi_Class value of Arabic_Number to make them appear in the same run as the numbers following them.

A few similar signs spanning numbers or letters are associated with scripts other than Arabic. See the discussion of U+070F SYRIAC ABBREVIATION MARK in *Section 9.3, Syriac*, and the discussion of U+110BD KAITHI NUMBER SIGN and U+110CD KAITHI NUMBER SIGN ABOVE in *Section 15.2, Kaithi*. All of these prefixed format controls, including the non-Arabic ones, are given the property value Prepended_Concatenation_Mark = True, to identify them as a class. They also have special behavior in text segmentation. (See Unicode Standard Annex #29, "Unicode Text Segmentation.")

U+0600 ARABIC NUMBER SIGN signals the beginning of a number. It is followed by a sequence of one or more Arabic digits and is rendered below the digits of the number. The length of its rendered display may vary with the number of digits. The sequence terminates with the occurrence of any non-digit character.

U+0601 ARABIC SIGN SANAH indicates a year (that is, as part of a date). This sign is also rendered below the digits of the number it precedes. Its appearance is a vestigial form of the Arabic word for *year*, /sanatu/ (*seen noon teh-marbuta*), but it is now a sign in its own right and is widely used to mark a numeric year even in non-Arabic languages where the Arabic word would not be known.

U+0602 ARABIC FOOTNOTE MARKER is a specialized variant of number sign. Its use indicates that the number so marked represents a footnote number in the text.

U+0603 ARABIC SIGN SAFHA is another specialized variant of number sign. It marks a page number.

U+0604 ARABIC SIGN SAMVAT is a specialized variant of date sign used specifically to write dates of the Śaka era. The shape of the glyph is a stylized abbreviation of the word *samvat*, the name of this calendar. It is seen in the Urdu orthography, where it contrasts with conventions used to display dates from the Gregorian or Islamic calendars.

U+0605 ARABIC NUMBER MARK ABOVE is a specialized variant of number sign. It is used in Arabic text with Coptic numbers, such as in early astronomical tables. Unlike the other Arabic number signs, it extends across the top of the sequence of digits, and is used with Coptic digits, rather than with Arabic digits. (See also the discussion of supralineation and the numerical use of letters in *Section 7.3, Coptic*.)

U+0890 ARABIC POUND MARK ABOVE and U+0891 ARABIC PIASTRE MARK ABOVE are Egyptian currency signs which extend across the top of a sequence of digits. The shape of the *pound mark* is usually based on a dotless head of *jeem* above the

amount. It is occasionally based on a dotted *jeem* instead. The shape of the *piastre mark* is written using a mirrored version of the pound mark. They are used in advertising and price tags, as well as in handwritten texts.

U+06DD ARABIC END OF AYAH is another sign used to span numbers, but its rendering is somewhat different. Rather than extending below the following digits, this sign *encloses* the digit sequence. This sign is used conventionally to indicate numbered verses in the Quran.

U+06DE ARABIC START OF RUB EL HIZB is an in-text marker. In printed Qurans, it appears in running text by itself, usually adjacent to an *end of ayah* marker. Although the original symbol for *rub el hizb* consists of octagonal overlayed squares, the actual glyph seen in various editions can be more ornate, as shown in the Unicode code charts. The *rub el hizb* indicates the boundaries of the parts of sections of the *hizb*. It can appear at the start or end of a section and is displayed without interaction with adjacent text.

U+08E2 ARABIC DISPUTED END OF AYAH is a specialized variant of the *end of ayah*. It is seen occasionally in Quranic text to mark a verse for which there is scholarly disagreement about the location of the end of the verse.

Because digit choice is dependent on regional use, these marks may be used with European digits (U+0030..U+0039), Arabic-Indic digits (U+0660..U+0669) or with extended Arabic-Indic digits (U+06F0..U+06F9). Implementations should support up to three or four digits. *Figure 9-7* shows examples of how these are formatted with varying numbers of digits. In these examples, each instance is separated by an arbitrary letter *hamza*, to help visualize how the formatted sequences interact with the Arabic baseline.

Figure 9-7. Arabic Signs Spanning Numbers

Character and Glyph		Sample
0600 NUMBER SIGN	⬚	ءﺎﻟﺀ ءﺎ١٢ﻟﺀ ء١٢٣ﻟﺀ
0601 SIGN SANAH (year sign)	⬚	ءﻟﺎﺀ ءﻟﻰ١٩ﺀ ءﻟﻰ١٩٨ﺀ ءﻟﻰ١٩٨٤ﺀ
0602 FOOTNOTE MARKER	⬚	ءﻟﺎﺀ ءﻟ١٢ﺀ
0603 SIGN SAFHA	⬚	ءﺻﺎﺀ ءﺻ١٢ﺀ ءﺻ١٢٣ﺀ ءﺻ١٢٣٤ﺀ
0604 SIGN SAMVAT	⬚	ءﺳﻤﺎﺀ ءﺳﻢ١٢ﺀ ءﺳﻢ١٢٣ﺀ ءﺳﻢ١٢٣٤ﺀ
0605 NUMBER MARK ABOVE	⬚	ءﺍﺀ ءﺍ١٢ﺀ ءﺍ١٢٣ﺀ ءﺍ١٢٣٤ﺀ
0890 POUND MARK ABOVE	⬚	ءﺍﺀ ءﺍ١٢ﺀ ءﺍ١٢٣ﺀ ءﺍ١٢٣٤ﺀ
0891 PIASTRE MARK ABOVE	⬚	ءﺍﺀ ءﺍ١٢ﺀ ءﺍ١٢٣ﺀ ءﺍ١٢٣٤ﺀ
06DD END OF AYAH	⬚	ءﺍﺀ ءﺍ١٢ﺀ ءﺍ١٢٣ﺀ
08E2 DISPUTED END OF AYAH	⬚	ءﺍﺀ ءﺍ١٢ﺀ ءﺍ١٢٣ﺀ

Poetic Verse Sign. U+060E ARABIC POETIC VERSE SIGN is a special symbol often used to mark the beginning of a poetic verse. Although it is similar to U+0602 ARABIC FOOTNOTE MARKER in appearance, the poetic sign is simply a symbol. In contrast, the footnote marker is a format control character that has complex rendering in conjunction with following digits. U+060F ARABIC SIGN MISRA is another symbol used in poetry.

9.2.2 Arabic Cursive Joining

Minimum Rendering Requirements. A rendering or display process must convert between the logical order in which characters are placed in the backing store and the visual (or physical) order required by the display device. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," for a description of the conversion between logical and visual orders.

The cursive nature of the Arabic script imposes special requirements on display or rendering processes that are not typically found in Latin script-based systems. At a minimum, a display process must select an appropriate glyph to depict each Arabic letter according to its immediate *joining* context; furthermore, in almost every font

style, it must substitute certain ligature glyphs for sequences of Arabic characters. The remainder of this section specifies a minimum set of rules that provide legible Arabic joining and ligature substitution behavior.

Joining Types. Each Arabic letter must be depicted by one of a number of possible contextual glyph forms. The appropriate form is determined on the basis of the cursive joining behavior of that character as it interacts with the cursive joining behavior of adjacent characters. In the Unicode Standard, such cursive joining behavior is formally described in terms of values of a character property called Joining_Type. Each Arabic character falls into one of the types shown in *Table 9-3*. (See ArabicShaping.txt in the Unicode Character Database for a complete list.)

Table 9-3. Primary Arabic Joining Types

Joining_Type	Examples and Comments
Right_Joining (R)	ALEF, DAL, THAL, REH, ZAIN...
Left_Joining (L)	None (in Arabic)
Dual_Joining (D)	BEH, TEH, THEH, JEEM...
Join_Causing (C)	U+200D ZERO WIDTH JOINER and U+0640 ARABIC TATWEEL. These characters are distinguished from the dual-joining characters in that they do not change shape themselves.
Non_Joining (U)	U+200C ZERO WIDTH NON-JOINER and all spacing characters, except those explicitly mentioned as being one of the other joining types, are non-joining. These include U+0621 ARABIC LETTER HAMZA, U+0674 ARABIC LETTER HIGH HAMZA, spaces, digits, punctuation, non-Arabic letters, and so on. Also, U+0600 ARABIC NUMBER SIGN..U+0605 ARABIC NUMBER MARK ABOVE and U+06DD ARABIC END OF AYAH.
Transparent (T)	All nonspacing marks (General Category Mn or Me) and most format control characters (General Category Cf) are transparent to cursive joining. These include U+064B ARABIC FATHATAN and other Arabic *tashkil*, U+0655 ARABIC HAMZA BELOW, U+0670 ARABIC LETTER SUPERSCRIPT ALEF, combining Quranic annotation signs, and nonspacing marks from other scripts. Also U+070F SYRIAC ABBREVIATION MARK.

In *Table 9-3*, *right* and *left* refer to visual order, so a Joining_Type value of Right_Joining indicates that a character cursively joins to a character displayed to its right in visual order. (For a discussion of the meaning of Joining_Type values in the context of a vertically rendered script, see "Cursive Joining" in *Section 14.4, Phags-pa*.) The Arabic characters with Joining_Type = Right_Joining are exemplified in more detail in *Table 9-8*, and those with Joining_Type = Dual_Joining are shown in *Table 9-7*. When characters do not join or cause joining (such as DAMMA), they are classified as transparent.

The Phags-pa and Manichaean scripts have a few Left_Joining characters, which are otherwise unattested in the Arabic and Syriac scripts. See *Section 10.5, Manichaean*. For a discussion of the meaning of Joining_Type values in the context of a vertically rendered script, see "Cursive Joining" in *Section 14.4, Phags-pa*.

Table 9-4 defines derived superclasses of the primary Arabic joining types; those derived types are used in the cursive joining rules. In this table, *right* and *left* refer to visual order.

Table 9-4. Derived Arabic Joining Types

Description	Derivation
Right join-causing	Superset of dual-joining, left-joining, and join-causing
Left join-causing	Superset of dual-joining, right-joining, and join-causing

Joining Rules. The following rules describe the joining behavior of Arabic letters in terms of their display (visual) order. In other words, the positions of letterforms in the included examples are presented as they would appear on the screen *after* the Bidirectional Algorithm has reordered the characters of a line of text.

An implementation may choose to restate the following rules according to logical order so as to apply them *before* the Bidirectional Algorithm's reordering phase. In this case, the words *right* and *left* as used in this section would become *preceding* and *following*.

In the following rules, if X refers to a character, then various glyph types representing that character are referred to as shown in *Table 9-5*.

Table 9-5. Arabic Glyph Types

Glyph Type	Description
X_n	Non-joining glyph form that does not join on either side.
X_r	Right-joining glyph form (both right-joining and dual-joining characters may employ this form)
X_l	Left-joining glyph form (both left-joining and dual-joining characters may employ this form)
X_m	Dual-joining (medial) glyph form that joins on both left and right (only dual-joining characters employ this form)

R1 *Transparent characters do not affect the joining behavior of base (spacing) characters. For example:*

$$\text{MEEM}_n + \text{SHADDA}_n + \text{LAM}_n \rightarrow \text{MEEM}_r + \text{SHADDA}_n + \text{LAM}_l$$

$$\text{م} + \ddot{\circ} + \text{ل} \rightarrow \text{م} + \ddot{\circ} + \text{ا} \rightarrow \text{لّم}$$

R2 *A right-joining character X that has a right join-causing character on the right will adopt the form X_r . For example:*

$$\text{ALEF}_n + \text{TATWEEL}_n \rightarrow \text{ALEF}_r + \text{TATWEEL}_n$$

$$\text{ا} + \_ \rightarrow \text{ا} + \_ \rightarrow \text{ل}$$

R3 *A left-joining character X that has a left join-causing character on the left will adopt the form X_l.*

R4 *A dual-joining character X that has a right join-causing character on the right and a left join-causing character on the left will adopt the form X_m. For example:*

$$\text{TATWEEL}_n + \text{MEEM}_n + \text{TATWEEL}_n \rightarrow \text{TATWEEL}_n + \text{MEEM}_m + \text{TATWEEL}_n$$

$$\_ + \text{م} + \_ \rightarrow \_ + \text{ﻤ} + \_ \rightarrow \text{ﻤ}$$

R5 *A dual-joining character X that has a right join-causing character on the right and no left join-causing character on the left will adopt the form X_r .For example:*

$$\text{MEEM}_n + \text{TATWEEL}_n \rightarrow \text{MEEM}_r + \text{TATWEEL}_n$$

$$\text{م} + \_ \rightarrow \text{م} + \_ \rightarrow \text{ﻢ}$$

R6 *A dual-joining character X that has a left join-causing character on the left and no right join-causing character on the right will adopt the form X_l.For example:*

$$\text{TATWEEL}_n + \text{MEEM}_n \rightarrow \text{TATWEEL}_n + \text{MEEM}_l$$

$$\_ + \text{م} \rightarrow \_ + \text{ﻣ} \rightarrow \text{ﻣ}$$

R7 *If none of the preceding rules applies to a character X, then it will adopt the non-joining form X_n.*

The cursive joining behavior described here for the Arabic script is also generally applicable to other cursive scripts such as Syriac. Specific circumstances may modify the application of the rules just described.

As noted earlier in this section, the ZERO WIDTH NON-JOINER may be used to prevent joining, as in the Persian plural suffix or Ottoman Turkish vowels.

9.2.3 Arabic Ligatures

Ligature Classes. The *lam-alef* type of ligatures are extremely common in the Arabic script. These ligatures occur in almost all font designs, except for a few modern styles. When supported by the style of the font, *lam-alef* ligatures are considered obligatory. This means that all character sequences rendered in that font, which match the rules specified in the following discussion, must form these ligatures.

In the majority of styles used for writing the Arabic script, including the predominant *Hafs* style, the *lam* is the right part of the ligature, and the *alef* is the left part of the ligature. However, in the *al-Dani* style of writing Arabic script, common in northern Africa, the practice is reversed: the *alef* is the right part and *lam* is the left part. This difference in the styles of writing Arabic is important for font developers to understand. Logical order would still be used in both styles: this means that in the *al-Dani* style of *lam-alef,* marks are positioned differently on the *lam-alef* ligature. See *Figure 9-8* for a comparison of rendering the sequence <*lam*, *sukun*, *alef-with-hamza-above*, *damma*> in the two styles mentioned.

Figure 9-8. *Lam-alef* with Marks

Hafs (*alef* is left part of ligature)	al-Dani (*alef* is right part of ligature)

The important thing to note in this figure is the placement of the marks over the parts of the ligature. The exact shapes of the ligature and the marks depend on the style in use.

In general, the *lam-alef* ligature will be formed by any character in the LAM joining group followed by any character from the ALEF joining group. Many *lam-alef* combi-

nations with the specialized *alef* additions in the range U+0870..U+0882 are not attested in actual practice. In such cases, the *lam-alef* ligature should not be considered obligatory.

Many other Arabic ligatures are discretionary. Their use depends on the font design.

For the purpose of describing the obligatory Arabic ligatures, certain characters are subject to the same requirements as *lam* and *alef*. As described in the text that follows, these fall into the joining groups LAM and ALEF, respectively. Examples of these include U+0644 ARABIC LETTER LAM, U+06B5 ARABIC LETTER LAM WITH SMALL V, U+0623 ARABIC LETTER ALEF WITH HAMZA ABOVE, and U+0622 ARABIC LETTER ALEF WITH MADDA ABOVE. The complete list is available in Arabic-Shaping.txt in the Unicode Character Database.

Ligature Rules. The following rules describe the formation of obligatory ligatures. They are applied after the preceding joining rules. As for the joining rules just discussed, the following rules describe ligature behavior of Arabic letters in terms of their display (visual) order.

In the ligature rules, if X and Y refer to characters, then various glyph types representing combinations of these characters are referred to as shown in *Table 9-6*.

Table 9-6. Arabic Ligature Notation

Symbol	Description
$(X\text{-}Y)_n$	Nominal ligature glyph form representing a combination of an X_r form and a Y_l form
$(X\text{-}Y)_r$	Right-joining ligature glyph form representing a combination of an X_r form and a Y_m form

L1 ***Transparent characters do not affect the ligating behavior of base (nontransparent) characters. For example:***

$$\text{ALEF}_r + \text{FATHA}_n + \text{LAM}_l \rightarrow (\text{LAM-ALEF})_n + \text{FATHA}_n$$

L2 ***Any sequence with ALEF$_r$ on the left and LAM$_m$ on the right will form the ligature (LAM-ALEF)$_r$. For example:***

$$\text{l} + \text{l} \rightarrow \text{ﻼ} \qquad (\text{not} \; \text{ll})$$

L3 ***Any sequence with ALEF$_r$ on the left and LAM$_l$ on the right will form the ligature (LAM-ALEF)$_n$. For example:***

$$\text{l} + \text{l} \rightarrow \text{ﻻ} \qquad (\text{not} \; \text{ll})$$

Optional Features. Many other ligatures and contextual forms are optional, depending on the font and application. Some of these presentation forms are encoded in the ranges U+FB50..U+FDFF and U+FE70..U+FEFE. However, these forms should *not* be used in general interchange. Moreover, it is not expected that every Arabic font will contain all of these forms, nor that these forms will include all presentation forms used by every font.

More sophisticated rendering systems will use additional shaping and placement. For example, contextual placement of the nonspacing vowels such as *fatha* will provide better appearance. The justification of Arabic tends to stretch words instead of adding width to spaces. Basic stretching can be done by inserting *tatweel* between characters shaped by rules R2, R4, R5, R6, L2, and L3; the best places for inserting tatweel will depend on the font and rendering software. More powerful systems will choose different shapes for characters such as *kaf* to fill the space in justification.

9.2.4 Arabic Joining Groups

The Arabic characters with the property values Joining_Type = Dual_Joining and Joining_Type = Right_Joining can each be subdivided into shaping groups, based on the behavior of their letter skeletons when shaped in context. The Unicode character property that specifies these groups is called Joining_Group.

The Joining_Type and Joining_Group values for all Arabic characters are explicitly specified in ArabicShaping.txt in the Unicode Character Database. For convenience in reference, the Joining_Type values are extracted and listed in DerivedJoining-Type.txt and the Joining_Group values are extracted and listed in DerivedJoining-Group.txt.

Dual-Joining. <u>Table 9-7</u> exemplifies dual-joining Arabic characters and illustrates the forms taken by the letter skeletons and their *ijam* marks in context. Dual-joining characters have four distinct forms, for isolated, final, medial, and initial contexts, respectively. The name for each joining group is based on the name of a representative letter that is used to illustrate the shaping behavior. All other Arabic characters are merely variations on these basic shapes, with diacritics added, removed, moved, or replaced. For instance, the BEH joining group applies not only to U+0628 ARABIC LETTER BEH, which has a single dot below the skeleton, but also to U+062A ARABIC LETTER TEH, which has two dots above the skeleton, and to U+062B ARABIC LETTER THEH, which has three dots above the skeleton, as well as to the Persian and Urdu letter U+067E ARABIC LETTER PEH, which has three dots below the skeleton. The joining groups in the table are organized by shape and not by standard Arabic alphabetical order. Note that characters in some joining groups have dots in some contextual forms, but not others, or their dots may move to a different position. These joining groups include NYA, FARSI YEH, and BURUSHASKI YEH BARREE.

Table 9-7. Dual-Joining Arabic Characters

Joining Group	X_n	X_r	X_m	X_l	Notes
BEH	ب	ب	بـ	بـ	Includes TEH and THEH.
NOON	ن	ن	نـ	نـ	Includes NOON GHUNNA.
AFRICAN NOON	ں	ں	نـ	نـ	
NYA	ث	ث	پـ	پـ	Jawi NYA.
YEH	ي	ي	يـ	يـ	Includes ALEF MAKSURA.
FARSI YEH	ى	ى	يـ	يـ	
KASHMIRI YEH	کی	ى	ؠـ	ؠـ	
THIN YEH			ؠـ	ؠـ	Final and isolated forms are not attested.
BURUSHASKI YEH BARREE	ےٚ	ےٚ	ؠٚـ	ؠٚـ	Dual joining, as opposed to YEH BARREE.
HAH	ح	ح	حـ	حـ	Includes KHAH and JEEM.
SEEN	س	س	سـ	سـ	Includes SHEEN.
SAD	ص	ص	صـ	صـ	Includes DAD.
TAH	ط	ط	ط	ط	Includes ZAH.
AIN	ع	ع	عـ	عـ	Includes GHAIN.
FEH	ف	ف	فـ	فـ	
AFRICAN FEH	ڢ	ڢ	ڢـ	ڢـ	
QAF	ق	ق	قـ	قـ	
AFRICAN QAF	ڧ	ڧ	ڧـ	ڧـ	
MEEM	م	م	مـ	مـ	
HEH	ه	ه	هـ	هـ	
KNOTTED HEH	ھ	ھ	ھـ	ھـ	See *Table 9-9* for more information on regional variation.
HEH GOAL	ہ	ہ	ہـ	ہـ	Includes HAMZA ON HEH GOAL.

KAF	ك	ك	ک	ک
SWASH KAF	ک	ک	ک	ک
GAF	گ	گ	گ	گ	Includes KEHEH.
LAM	ل	ل	ﻟ	ﻟ

Right-Joining. *Table 9-8* exemplifies right-joining Arabic characters, illustrating the forms they take in context. Right-joining characters have only two distinct forms, for isolated and final contexts, respectively.

Table 9-8. Right-Joining Arabic Characters

Joining Group	X_n	X_r	Notes
ALEF	ا	ا	
WAW	و	و	
STRAIGHT WAW	ٯ	ٯ	Tatar STRAIGHT WAW.
DAL	د	د	Includes THAL.
REH	ر	ر	Includes ZAIN.
TEH MARBUTA	ة	ة	Includes HAMZA ON HEH.
TEH MARBUTA GOAL	ة	ٔ	
YEH WITH TAIL	ى	ى	
YEH BARREE	ے	ے	
ROHINGYA YEH	ﻳ	ﻳ	

Some characters occur only at the end of words or morphemes when correctly spelled; these are called *trailing characters*. Examples include TEH MARBUTA and DAMMATAN. When trailing characters are joining (such as TEH MARBUTA), they are classified as right-joining, even when similarly shaped characters are dual-joining. Other characters, such as ALEF MAKSURA, are considered trailing in modern Arabic, but are dual-joining in Quranic Arabic and languages like Uyghur. These are classified as dual-joining.

Letter heh. In the case of U+0647 ARABIC LETTER HEH, the glyph ﻫ is shown in the code charts. This form is often used to reduce the chance of misidentifying *heh* as U+0665 ARABIC-INDIC DIGIT FIVE, which has a very similar shape. The isolated

forms of U+0647 ARABIC LETTER HEH and U+06C1 ARABIC LETTER HEH GOAL both look like U+06D5 ARABIC LETTER AE.

U+06BE ARABIC LETTER HEH DOACHASHMEE is used to represent any *heh*-like letter that appears with left stems in all contextual forms. All four forms should have two horizontal or vertical "eyes." The exact contextual shapes of the letter depend on the language and the style of writing. Four variations for KNOTTED HEH are shown in *Table 9-9*.

Table 9-9. Letter *heh* Shapes

Code Points	Name	Joining Group	X_n	X_r	X_m	X_l	Notes
0647 FEE9..FEEC	HEH	HEH	ه	ﻪ	ﻬ	ﻫ	Standard forms
06BE FBAA..FBAD	HEH DOACHASHMEE	KNOTTED HEH	ھ	ﻬ	ﻬ	ﻫ	Standard forms, Uighur, Kazakh
			ھ	ﮭ	ﻬ	ﻫ	Behdini Kurdish
			ھ	ﮭ	ﮭ	ﻫ	Possibly used in Sindhi
			ﮭ	ﮬ	ﮬ	ﮪ	Nastaliq

Letter yeh. There are many complications in the shaping of the Arabic letter *yeh*. These complications have led to the encoding of several different characters for *yeh* in the Unicode Standard, as well as the definition of several different joining groups involving *yeh*. The relationships between those characters and joining groups for *yeh* are explained here.

U+06CC ARABIC LETTER FARSI YEH is used in Persian, Urdu, Pashto, Azerbaijani, Kurdish, and various minority languages written in the Arabic script, and also Quranic Arabic. It behaves differently from most Arabic letters, in a way surprising to some native Arabic language speakers. The letter has two horizontal dots below the skeleton in initial and medial forms, but no dots in final and isolated forms. Compared to the two Arabic language *yeh* forms, FARSI YEH is exactly like U+0649 ARABIC LETTER ALEF MAKSURA in final and isolated forms, but exactly like U+064A ARABIC LETTER YEH in initial and medial forms, as shown in *Table 9-10*. However, U+06CC ARABIC LETTER FARSI YEH followed by U+0654 ARABIC HAMZA ABOVE should retain its dots in initial and medial forms.

Table 9-10. Forms of the Arabic Letter *yeh*

Character	Joining Group	X_n	X_r	X_m	X_l
U+0649 ALEF MAKSURA	YEH	ى	ى	ﺪ	ﺪ

U+064A YEH	YEH	ي	ي	ﻴ	ﻳ
U+06CC FARSI YEH	FARSI YEH	ى	ى	ﻴ	ﻳ
U+0886 THIN YEH	THIN YEH			ﻴ	ﻳ
U+0777 YEH WITH DIGIT FOUR BELOW	YEH	ﻱ	ﻱ	ﻴ	ﻳ
U+0620 KASHMIRI YEH	KASHMIRI YEH	ى	ى	ﻴ	ﻳ
U+06D2 YEH BARREE	YEH BARREE	ے	ے		
U+077A YEH BARREE WITH DIGIT TWO ABOVE	BURUSHASKI YEH BARREE	ﮯ	ﮯ	ﻴ	ﻳ
U+08AC ROHINGYA YEH	ROHINGYA YEH	ﮡ	ﮡ		

Other characters of the joining group FARSI YEH follow the same pattern. These YEH forms appear with two dots aligned horizontally below them in initial and medial forms, but with no dots below them in final and isolated forms. Characters with the joining group YEH behave in a different manner. Just as U+064A ARABIC LETTER YEH retains two dots below in all contextual forms, other characters in the joining group YEH retain whatever mark appears below their isolated form in all other contexts. For example, U+0777 ARABIC LETTER FARSI YEH WITH EXTENDED ARABIC-INDIC DIGIT FOUR BELOW carries an Urdu-style digit four as a diacritic below the *yeh* skeleton, and retains that diacritic in all positions, as shown in the fourth row of *Table 9-10*. Note that the joining group cannot always be derived from the character name alone. The complete list of characters with the joining group YEH or FARSI YEH is available in ArabicShaping.txt in the Unicode Character Database.

In the orthographies of Arabic and Persian, the *yeh barree* has always been treated as a stylistic variant of *yeh* in final and isolated positions. When the Perso-Arabic writing system was adapted and extended for use with the Urdu language, *yeh barree* was adopted as a distinct letter to accommodate the richer vowel repertoire of Urdu. South Asian languages such as Urdu and Kashmiri use *yeh barree* to represent the /e/ vowel. This contrasts with the /i/ vowel, which is usually represented in those languages by U+06CC ARABIC LETTER FARSI YEH. The encoded character U+06D2 ARABIC LETTER YEH BARREE is classified as a right-joining character, as shown in *Table 9-10*. On that basis, when the /e/ vowel needs to be represented in initial or medial positions with a *yeh* shape in such languages, one should use U+06CC ARABIC LETTER FARSI YEH. In the unusual circumstances where one wishes to distinctly represent the /e/ vowel in word-initial or word-medial positions, a higher level protocol should be used.

For the Burushaski language, two characters that take the form of *yeh barree* with a diacritic, U+077A ARABIC LETTER YEH BARREE WITH EXTENDED ARABIC-INDIC

DIGIT TWO ABOVE and U+077B ARABIC LETTER YEH BARREE WITH EXTENDED ARABIC-INDIC DIGIT THREE ABOVE, are classified as dual-joining. These characters have a separate joining group called BURUSHASKI YEH BARREE, as shown for U+077A in the last row of *Table 9-10*.

U+0620 ARABIC LETTER KASHMIRI YEH is used in Kashmiri text to indicate that the preceding consonantal sound is palatalized. The letter has the form of a *yeh* with a diacritic small circle below in initial and medial forms, but its final and isolated forms appear as truncated *yeh* shapes (ی) without the diacritic ring. It has a joining group of its own, KASHMIRI YEH, with the shapes as shown in *Table 9-10*, as well as *Table 9-7*. (Prior to Version 16.0, the Unicode Standard had specified that when written in the Naskh style, the letter had different shapes than when written in Nastaliq style; that specification was erroneous.)

U+08AC ARABIC LETTER ROHINGYA YEH is used in the Arabic orthography for the Rohingya language of Myanmar. It represents a *medial ya*, corresponding to the use of U+103B MYANMAR CONSONANT SIGN MEDIAL YA in the Myanmar script. It is a right-joining letter. It only occurs after certain consonants, forming a conjunct letter with those consonants.

U+0626 ARABIC LETTER YEH WITH HAMZA ABOVE normally has the *hamza* positioned over the bowl of the glyph in isolate and final forms. For the Kyrgyz language the *hamza* is positioned at the top right of the glyph in isolate and final forms, as shown in *Table 9-11*.

Table 9-11. Glyph Variation for U+0626 *Yeh with Hamza Above*

	X_n	X_r	X_m	X_l
Standard	ئ	ئ	ﺌ	ﺋ
Kyrgyz-style	ئ	ئ	ﺌ	ﺋ

Noon Ghunna. The letter *noon ghunna* is used to mark nasalized vowels at the ends of words and some morphemes in Urdu, Balochi, and other languages of Southern Asia. It is represented by U+06BA ARABIC LETTER NOON GHUNNA. The *noon ghunna* has the shape of a dotless *noon* and typically appears only in final and isolated contexts in these languages. In the middle of words and morphemes, the normal *noon,* U+0646 ARABIC LETTER NOON, is used instead. To avoid ambiguity, sometimes a special mark, U+0658 ARABIC MARK NOON GHUNNA, is added to the dotted *noon* to indicate nasalization.

U+06BA ARABIC LETTER NOON GHUNNA is also used as a dotless *noon* for the *noon* skeleton in all four of its contextual forms. As such, it is used in representation of early Arabic and Quranic Arabic texts. Rendering systems should display U+06BA as

a dual-joining letter, with all four contextual forms shown dotless, regardless of the language of the text.

Advanced text entry applications for Urdu, Balochi, and other languages using *noon ghunna* may include specialized logic for its handling. For example, they might detect mid-word usage of the *noon ghunna* key and emit the regular dotted *noon* character (U+0646) instead, as appropriate for spelling in that context.

Letters for Warsh. There is a set of widespread orthographic conventions for Arabic writing in West and Northwest Africa known as *Warsh*. Among other differences from the better-known *Hafs* orthography of the Middle East, there are significant differences in Warsh regarding the placement of *ijam* dots on several important Arabic letters. Several "African" letters are encoded in the Arabic Extended-A block specifically to account for these differences in dot placement.

The specialized letters for Warsh share the skeleton with the corresponding, regular Arabic letters. However, they differ in the placement of dots. The Warsh letters have no dots in final or isolated positional contexts. This is illustrated by U+08BD ARABIC LETTER AFRICAN NOON. Unlike U+0646 ARABIC LETTER NOON, which displays a dot above in *all* positional contexts, *african noon* displays a dot above in initial and medial position, and no dot in final or isolated position. This contrast can be clearly seen in *Table 9-7*.

U+08BB ARABIC LETTER AFRICAN FEH and U+08BC ARABIC LETTER AFRICAN QAF also lose all dots in final or isolated position, but exhibit a somewhat different pattern for initial and medial position. The basic skeletons for *feh* and for *qaf* are identical for those letters in initial and medial position. In the Hafs orthography, the *feh* takes a single dot above in all positions, while the *qaf* takes *two* dots above in all positions. The Warsh orthography distinguishes the two letters differently: the *feh* takes a single dot *below* in initial or medial position, while the *qaf* takes a single dot *above* in initial or medial position. These contextual differences in the placement of the dots for these letters can also be seen in *Table 9-7*.

Letters for Ajami. The Hausa and Wolof languages of West Africa use an Arabic-based orthography known as *ajami*. The *ajami* orthography contains additional, specialized letters with three dots above or below. This three dot *ijam* is known as a *wagaf* in Hausa. In the Kano/Maghribi Arabic style used in this region, the *wagaf* is noticeably smaller than any other *ijam* that may also occur on these specialized characters or other Arabic letters. For example, when rendered in the Kano/Maghribi style in a Hausa font, U+0751 ARABIC LETTER BEH WITH DOT BELOW AND THREE DOTS ABOVE will show the dot below in a dark, large size, while the three dots above for the *wagaf* are distinctly smaller. This distinction in size tends to be much less noticeable when the same letters are rendered using a standard, *naskh* style Arabic font, as shown in the code charts.

Joining Groups in Other Scripts. Other scripts besides Arabic also have cursive joining behavior and associated per-character values for Joining_Type and Joining_Group. Those values are also listed in ArabicShaping.txt in the Unicode Character Database, in sections devoted to each particular script. See the script descriptions for such scripts in the core specification—for example, Syriac and Manichaean—for detailed discussions of cursive joining behavior and tables of joining groups for those scripts.

For the Arabic script, Joining_Group values are assigned for each distinct letter skeleton in all instances—even for the small number of cases, such as *heh goal*, where only a single character is associated with that Joining_Group. This is appropriate for Arabic, because the script has cosmopolitan use, and many letters have been modified with various *nukta* diacritics to form new letters for non-Arabic languages using the script. This pattern of comprehensive assignment of Joining_Group values to all letter skeletons also applies for the Syriac and Manichaean scripts.

For other cursive joining scripts with less well-defined joining groups, all letters are simply assigned the value No_Joining_Group. This does not necessarily mean that no identifiable letter skeletons occur, but rather that no complete analysis has been done that would indicate more than one letterform uses a shared skeleton for cursive joining. Examples include: Mongolian, Phags-Pa, Psalter Pahlavi, Sogdian, and Adlam.

Starting with Unicode 11.0, even in cases where a newly encoded script with cursive joining behavior includes *some* characters which share letter skeletons, most characters are given the No_Joining_Group value. This applies, for example, to the Hanifi Rohingya script, which has a few explicit Joining_Group values, but for which all other characters have the No_Joining_Group value.

In the future, characters with the No_Joining_Group value in scripts with cursive joining behavior may end up being given explicit new Joining_Group values, where further analysis clearly demonstrates use of shared skeletons in cursive joining, or where new, diacritically modified letters are added to the encoding for that script.

9.2.5 Combining Hamza

Combining Hamza Above. U+0654 ARABIC HAMZA ABOVE is intended both for the representation of *hamza* semantics in combination with certain Arabic letters, and as a diacritical mark occasionally used in combinations to derive extended Arabic letters. There are a number of complications regarding its use, which interact with the rules for the rendering of Arabic letter *yeh* and which result from the need to keep Unicode normalization stable.

U+0654 ARABIC HAMZA ABOVE should not be used with U+0649 ARABIC LETTER ALEF MAKSURA. Instead, the precomposed U+0626 ARABIC LETTER YEH WITH HAMZA ABOVE should be used to represent a *yeh*-shaped base with no dots in any positional form, and with a *hamza* above. Because U+0626 is canonically equivalent to the sequence <U+064A ARABIC LETTER YEH, U+0654 ARABIC HAMZA ABOVE>, when U+0654 is applied to U+064A ARABIC LETTER YEH, the *yeh* should lose its dots in all positional forms, even though *yeh* retains its dots when combined with other marks.

A separate, non-decomposable character, U+08A8 ARABIC LETTER YEH WITH TWO DOTS BELOW AND HAMZA ABOVE, is used to represent a *yeh*-shaped base with a *hamza* above, but with retention of dots in all positions. This letter is used in the Fulfulde language in Cameroon, to represent a palatal implosive.

In most other cases when a *hamza* is needed as a mark above for an Arabic letter, U+0654 ARABIC HAMZA ABOVE can be freely used in combination with basic Arabic letters. Three exceptions are the extended Arabic letters U+0681 ARABIC LETTER HAH WITH HAMZA ABOVE, U+076C ARABIC LETTER REH WITH HAMZA ABOVE, and U+08A1 ARABIC LETTER BEH WITH HAMZA ABOVE, where the *hamza* mark is functioning as an *ijam* (diacritic), rather than as a normal *hamza*. In those three cases, the extended Arabic letters have no canonical decompositions; consequently, the preference is to use those precomposed forms, rather than applying U+0654 ARABIC HAMZA ABOVE to *hah*, *reh*, or *beh* respectively.

In Persian and Urdu, a *hamza above* is frequently used for the *ezafe* sound /je/. This should be represented using U+0654 ARABIC HAMZA ABOVE after the *heh* letter appropriate for the orthography, as opposed to the precomposed U+06C0 which decomposes to a *heh* form not used in Persian and Urdu.

In Kashmiri, a *hamza above* is used as a vowel to represent the sound /ə/ over various different letters. In cases where it appears over a *beh*, *hah*, or *reh*, the precomposed letters U+0681, U+076C, and U+08A1 mentioned above should *not* be used. Instead, such Kashmiri text must be represented using *beh*, *hah*, or *reh* followed by U+0654 ARABIC HAMZA ABOVE.

These interactions between various letters and the *hamza* are summarized in *Table 9-12*.

Table 9-12. Arabic Letters With Hamza Above

Code Point	Name	Decomposition
0623	alef with hamza above	0627 0654
0624	waw with hamza above	0648 0654
0626	yeh with hamza above	064A 0654
06C2	heh goal with hamza above	06C1 0654

06D3	yeh barree with hamza above	06D2 0654
0681	hah with hamza above	None
076C	reh with hamza above	None
08A1	beh with hamza above	None
08A8	yeh with 2 dots below and hamza above	None

The first five entries in *Table 9-12* show the cases where the *hamza above* can be freely used, and where there is a canonical equivalence to the precomposed characters. The last four entries show the exceptions, where use of the *hamza above* is inappropriate, and where only the precomposed characters should be used.

High Hamza. The characters U+0675 ٵ ARABIC LETTER HIGH HAMZA ALEF, U+0676 ٶ ARABIC LETTER HIGH HAMZA WAW, U+0677 ٷ ARABIC LETTER U WITH HAMZA ABOVE, and U+0678 ٸ ARABIC LETTER HIGH HAMZA YEH are not recommended for use. Their compatibility decompositions are anomalous: the decomposed sequences are pairs of letters with right-to-left bidirectional character properties, with U+0674 ٴ ARABIC LETTER HIGH HAMZA as the second letter. When the decomposed sequences are processed using the Unicode Bidirectional Algorithm, the *hamza* will appear to the left of the other letter, whereas in the composite characters the *hamza* appears on the right. Thus, the ordering of characters in the decomposition sequences are the reverse of what is expected. Accordingly, appropriately-ordered pairs of letters beginning with U+0674 ٴ ARABIC LETTER HIGH HAMZA should be used instead. For example, the sequence <U+0674, U+0627> should be used rather than U+0675 ٵ ARABIC LETTER HIGH HAMZA ALEF. To facilitate correct text entry, input methods should be configured to generate the corresponding pairs of letters beginning with U+0674 ٴ ARABIC LETTER HIGH HAMZA.

Use of these characters in identifier systems can be problematic and can present a potential security risk. For example, IDNA 2003 permits U+0675 ٵ ARABIC LETTER HIGH HAMZA ALEF to be used in a domain name, but requires that to be mapped to the compatibility decomposed sequence <U+0627, U+0674> before conversion to punycode. However, the sequence <U+0674, U+0627> could also be used in a domain name. Two domain names that differ only in using U+0675 versus <U+0627, U+0674> would map to distinct punycode sequences but would be visually identical. Under IDNA 2008, the four composed characters (U+0675..U+0678) would no longer be permitted in a registered domain name, but applications can still accept them and map them into punycode, so risks from ambiguity still exist.

Malay Jawi uses U+0674 ٴ ARABIC LETTER HIGH HAMZA. In Jawi, the letter is the same size as U+0621 ء ARABIC LETTER HAMZA; however, unlike U+0621, it is positioned above the baseline at three-quarters height of the U+0627 ا ARABIC LETTER ALEF. Font designers can use language tagging in order to support the preferred shapes for both Kazakh and Jawi in multilingual fonts.

Quranic Texts. Most traditions of writing the Quran keep the skeleton of words intact from earlier Quranic manuscripts, but add dots and diacritics, including *hamzas*. Thus, words spelled with the medial form of U+0626 ARABIC LETTER YEH WITH HAMZA ABOVE in modern Arabic orthographies may appear in Quranic texts without the tooth typical of the letter. There is usually an elongation under the *hamza*, and the *hamza* may carry other diacritical marks, such as a *fatha*. This convention can be thought of as a modified version of *yeh-hamza*, and is represented with the sequence <U+0640 ARABIC TATWEEL, U+0654 ARABIC HAMZA ABOVE>. For example, in some Quranic traditions the word *yas'aluka* is represented by the sequence <*yeh*, *fatha*, *seen*, *sukun*, *tatweel*, *hamza above*, *fatha*, *lam*, *damma*, *kaf*, *fatha*>.

9.2.6 Other Letters for Extended Arabic

Jawi. U+06BD ARABIC LETTER NOON WITH THREE DOTS ABOVE is used for Jawi, which is Malay written using the Arabic script. Malay users know the character as *Jawi Nya*. Contrary to what is suggested by its Unicode character name, U+06BD displays with the three dots *below* the letter pointing downward when it is in the initial or medial position, making it look exactly like the initial and medial forms of U+067E ARABIC LETTER PEH. This is done to avoid confusion with U+062B ARABIC LETTER THEH, which appears in words of Arabic origin, and which has the same base letter shapes in initial or medial position, but with three dots above in all positions.

Kurdish. The Kurdish language is written in several different orthographies, which use either the Latin, Cyrillic, or Arabic scripts. When written using the Arabic script, Kurdish uses a number of extended Arabic letters, for an alphabet known as Soraní. Some of those extensions are shared with Persian, Urdu, or other languages: for example, U+06C6 ARABIC LETTER OE, which represents the Kurdish vowel [o]. Soraní also makes other unusual adaptations of the Arabic script, including the use of a digraph *waw+waw* to represent the long Kurdish vowel [uː]. That digraph is represented by a sequence of two characters, <U+0648 ARABIC LETTER WAW, U+0648 ARABIC LETTER WAW>.

Among the extended Arabic characters used exclusively for Soraní are U+0695 ARABIC LETTER REH WITH SMALL V BELOW (for the Kurdish *trilled r*) and U+06B5 ARABIC LETTER LAM WITH SMALL V (for the Kurdish *velarized l*).

The Arabic block also includes several extended Arabic characters whose origin was to represent dialectal or other poorly attested alternative forms of the Soraní alphabet extensions. U+0692 ARABIC LETTER REH WITH SMALL V is a dialectal variant of U+0695 which places the *small v* diacritic above the letter rather than below it. U+0694 is another variant of U+0695. U+06B6 and U+06B7 are poorly attested variants of U+06B5, and U+06CA is a poorly attested variant of U+06C6. None of these

alternative forms is required (or desired) for a regular implementation of the Kurdish Soraní orthography.

Sindhi Meem. In general, the distinction between a long tail and a short tail is stylistic. However, Sindhi specifically prefers the *meem* to have a short tail in isolate and final positions, as shown in *Table 9-13*.

Table 9-13. Glyph Variation for U+0645 *Meem*

	X_n	X_r	X_m	X_l
Standard	م	ﻤ	ﻢ	ﻣ
Sindhi-style	ﻡ	ﻤ	ﻢ	ﻣ

9.2.7 Arabic Supplement: U+0750–U+077F

The Arabic Supplement block contains additional extended Arabic letters for the languages used in Northern and Western Africa, such as Fulfulde, Hausa, Songhoy, and Wolof. In the second half of the 20th century, the use of the Arabic script was actively promoted for these languages. This block also contains a number of letters used for the Khowar, Torwali, and Burushaski languages, spoken primarily in Pakistan. Characters used for other languages are annotated in the character names list. Additional vowel marks used with these languages are found in the main Arabic block.

Marwari. U+076A ARABIC LETTER LAM WITH BAR is used to represent a flapped retroflexed lateral in the Marwari language in southern Pakistan. It has also been suggested for use in the Gawri language of northern Pakistan but it is unclear how widely it has been adopted there. Contextual shaping for this character is similar to that of U+0644 ARABIC LETTER LAM, including the requirement to form ligatures with characters of Joining_Group = ALEF.

9.2.8 Arabic Extended-A: U+08A0–U+08FF

The Arabic Extended-A block contains additional Arabic letters and vowel signs for use by a number of African languages from Chad, Senegal, Guinea, and Cameroon, and for languages of the Philippines. It also contains extended letters, vowel signs, and tone marks used by the Rohingya Fonna writing system for the Rohingya language in Myanmar, as well as several additional Quranic annotation signs. Characters used for other languages are annotated in the character names list.

One Quranic annotation sign, U+08D9 ◌ ARABIC SMALL LOW NOON WITH KASRA was given a mistaken Canonical_Combining_Class value when it was encoded in this block, and that value cannot be changed, due to normalization stability policies.

Section 5.8, "Workaround for Mistaken Canonical_Combining_Class Assignment" in Unicode Standard Annex #53, "Unicode Arabic Mark Rendering," provides more details about this character and explains how the Arabic Mark Transient Reordering Algorithm can be applied to get correct rendering behavior.

9.2.9 Arabic Extended-B: U+0870–U+089F

The Arabic Extended-B block comprises Quranic characters, especially those used in Northwest Africa, and characters from other orthographies, such as Bosnian and Pegon in Indonesia. The block also includes currency symbols and an abbreviation mark.

9.2.10 Arabic Extended-C: U+10EC0–U+10EFF

The Arabic Extended-C block comprises Quranic characters and characters from other orthographies, such as Pegon in Indonesia.

9.2.11 Arabic Presentation Forms-A: U+FB50–U+FDFF

This block contains a list of Arabic presentation forms encoded as characters primarily for compatibility reasons. These characters have a preferred representation that makes use of a normal (noncompatibility) Arabic character, or in many cases a sequence of Arabic characters. *Presentation form* is a mostly obsolete term for a contextually shaped glyph (for a single character) or for a ligature glyph (for a sequence of characters).

The presentation forms in this block consist of contextual (positional) variants of Extended Arabic letters, contextual variants of Arabic letter ligatures, spacing forms of Arabic diacritic combinations, contextual variants of certain Arabic letter/diacritic combinations, and Arabic phrase ligatures, including honorific word ligatures. The ligatures include a large set of presentation forms. However, the set of ligatures appropriate for any given Arabic font will generally not match this set precisely. Fonts will often include only a subset of these glyphs, and they may also include glyphs outside of this set. The included glyphs are generally not accessible as characters and are used only by rendering engines.

Ornate Parentheses. The alternative, ornate forms of parentheses (U+FD3E ORNATE LEFT PARENTHESIS and U+FD3F ORNATE RIGHT PARENTHESIS) for use with the Arabic script are considered traditional Arabic punctuation, rather than compatibility characters. These ornate parentheses are exceptional in rendering in bidirectional text; for legacy reasons, they do not have the Bidi_Mirrored property. Thus, unlike other parentheses, they do not automatically mirror when rendered in a bidirectional context.

Nuktas. Various patterns of single or multiple dots or other small marks are used diacritically to extend the core Arabic set of letters to represent additional sounds in other languages written with the Arabic script. Such dot patterns are known as *ijam* or *nuktas*. In the Unicode Standard, extended Arabic characters with nuktas are simply encoded as fully-formed base characters. However, there is an occasional need in pedagogical materials about the Arabic script to exhibit the various nuktas in isolation. The range of characters U+FBB2..U+FBC1 provides a set of symbols for this purpose. These are ordinary, spacing symbols with right-to-left directionality. They are *not* combining marks, and are not intended for the construction of new Arabic letters by use in combining character sequences. The Arabic pedagogical symbols do not partake of any Arabic shaping behavior. Their Joining_Type is Non_Joining, so if used in juxtaposition with an Arabic letter skeleton, they will break the cursive connection and render after the letter, instead of above or below it.

For clarity in display, those with the names including the word "above" should have glyphs that render high above the baseline, and those with names including "below" should be at or below the baseline.

Word Ligatures. The signs and symbols encoded at U+FD40..U+FD4F, U+FDCF, and U+FDF0..U+FDFF are word ligatures sometimes treated as a unit. Most of them are encoded for compatibility with older character sets and are rarely used, except the following:

U+FDF2 ARABIC LIGATURE ALLAH ISOLATED FORM is a very common ligature, used to display the name of God. When the formation of the *allah* ligature is desired, the recommended way to represent the word would be <*alef*, *lam*, *lam*, *shadda*, *superscript alef*, *heh*> <0627, 0644, 0644, 0651, 0670, 0647>. In non-Arabic languages, other forms of *heh*, such as *heh goal* (U+06C1), may also form the ligature. Extra care should be taken not to form the ligature in the absence of the *shadda* and the *superscript alef*, as the sequences <*alef*, *lam*, *lam*, *heh*> and <*alef*, *lam*, *lam*, *shadda*, *heh*> exist in Persian and other languages with different meanings or pronunciations, where the formation of the ligature would be incorrect and inappropriate.

U+FDFA ARABIC LIGATURE SALLALLAHOU ALAYHE WASALLAM and U+FDFB ARABIC LIGATURE JALLAJALALOUHOU are honorifics, commonly used after the name of the prophet Muhammad or God. Other honorific ligatures include U+FD40..U+FD4F, U+FDCF, and U+FDFD..U+FDFF. Their usage is comparable to the honorifics found at U+0610..U+0613, except that these are spacing characters. The same characters are sometimes used by Muslims writing in other scripts such as Latin and Cyrillic.

U+FDFD ARABIC LIGATURE BISMILLAH AR-RAHMAN AR-RAHEEM is a special ligated form of the *basmala*, a common opening phrase used by Muslims. The ligature is written in a multitude of ways. Its usage is common in writings by Muslims in non-Arabic scripts, even more than the honorifics mentioned above. It can be

displayed as a unit above text in several different scripts, such as Bengali and Thaana. Unlike the other Arabic word ligatures, this character does not have a compatibility decomposition.

U+FDFC RIAL SIGN is a condensed version of the word *rial*, the Iranian currency. The character was invented by a typewriter standardization committee in 1973 and is encoded in the Unicode Standard as a compatibility character, as it continues to be specified in Iranian national standards for character sets and keyboard layouts, including ISIRI 9147:2007. Except for a short life during the typewriter era, it has not received widespread usage outside standards, as Iranians prefer to spell out the word as <*reh*, *farsi yeh*, *alef*,*lam*>.

9.2.12 Arabic Presentation Forms-B: U+FE70–U+FEFF

This block contains additional Arabic presentation forms consisting of spacing or *tatweel* forms of Arabic diacritics, contextual variants of primary Arabic letters, and some of the obligatory LAM-ALEF ligatures. They are included here for compatibility with preexisting standards and legacy implementations that use these forms as characters. Instead of these, letters from the Arabic block (U+0600..U+06FF) should be used for interchange. Implementations should handle contextual glyph shaping by rendering rules when accessing glyphs from fonts, rather than by encoding contextual shapes as characters.

Spacing and Tatweel Forms of Arabic Diacritics. For compatibility with certain implementations, a set of spacing forms of the Arabic diacritics is provided here. The tatweel forms are combinations of the joining connector tatweel and a diacritic.

Zero Width No-Break Space. This character (U+FEFF), which is not an Arabic presentation form, is described in <u>*Section 23.8, Specials*</u>.

9.3 Syriac

9.3.1 Syriac: U+0700–U+074F

Syriac Language. The Syriac language belongs to the Aramaic branch of the Semitic family of languages. The earliest datable Syriac writing dates from the year 6 CE. Syriac is the active liturgical language of many communities in the Middle East (Syrian Orthodox, Assyrian, Maronite, Syrian Catholic, and Chaldaean) and Southeast India (Syro-Malabar and Syro-Malankara). It is also the native language of a considerable population in these communities.

Syriac is divided into two dialects. West Syriac is used by the Syrian Orthodox, Maronites, and Syrian Catholics. East Syriac is used by the Assyrians (that is, Ancient Church of the East) and Chaldaeans. The two dialects are very similar and have almost no differences in grammar and vocabulary. They differ in pronunciation and use different dialectal forms of the Syriac script.

Languages Using the Syriac Script. A number of modern languages and dialects employ the Syriac script in one form or another. They include the following:

1. *Literary Syriac*. The primary usage of Syriac script.

2. *Neo-Aramaic dialects*. The Syriac script is widely used for modern Aramaic languages, next to Hebrew, Cyrillic, and Latin. A number of Eastern Modern Aramaic dialects known as *Swadaya* (also called vernacular Syriac, modern Syriac, modern Assyrian, and so on, and spoken mostly by the Assyrians and Chaldaeans of Iraq, Turkey, and Iran) and the Central Aramaic dialect, *Turoyo* (spoken mostly by the Syrian Orthodox of the Tur Abdin region in southeast Turkey), belong to this category of languages.

3. *Garshuni* (Arabic written in the Syriac script). It is currently used for writing Arabic liturgical texts by Syriac-speaking Christians. Garshuni employs the Arabic set of vowels and overstrike marks.

4. *Christian Palestinian Aramaic* (also known as Palestinian Syriac). This dialect is no longer spoken.

5. *Other languages*. The Syriac script was used in various historical periods for writing Armenian and some Persian dialects. Syriac speakers employed it for writing Arabic, Ottoman Turkish, and Malayalam. Six special characters used for Persian and Sogdian were added in Version 4.0 of the Unicode Standard.

Shaping. The Syriac script is cursive and has shaping rules that are similar to those for Arabic. The Unicode Standard does not include any presentation form characters for Syriac.

Directionality. The Syriac script is written from right to left. Conformant implementations of Syriac script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Syriac Type Styles. Syriac texts employ several type styles. Because all type styles use the same Syriac characters, even though their shapes vary to some extent, the Unicode Standard encodes only a single Syriac script.

1. *Estrangela type style.* Estrangela (a word derived from Greek *strongulos,* meaning "rounded") is the oldest type style. Ancient manuscripts use this writing style exclusively. Estrangela is used today in West and East Syriac texts for writing headers, titles, and subtitles. It is the current standard in writing Syriac texts in Western scholarship.

2. *Serto or West Syriac type style.* This type style is the most cursive of all Syriac type styles. It emerged around the eighth century and is used today in West Syriac texts, Turoyo (Central Neo-Aramaic), and Garshuni.

3. *East Syriac type style.* Its early features appear as early as the sixth century; it developed into its own type style by the twelfth or thirteenth century. This type style is used today for writing East Syriac texts as well as Swadaya (Eastern Neo-Aramaic). It is also used today in West Syriac texts for headers, titles, and subtitles alongside the Estrangela type style.

4. *Christian Palestinian Aramaic.* Manuscripts of this dialect employ a script that is akin to Estrangela. It can be considered a subcategory of Estrangela.

The Unicode Standard provides for usage of the type styles mentioned above. It also accommodates letters and diacritics used in Neo-Aramaic, Christian Palestinian Aramaic, Garshuni, Persian, and Sogdian languages. *Examples are supplied in the Serto type style, except where otherwise noted.*

Character Names. Character names follow the East Syriac convention for naming the letters of the alphabet. Diacritical points use a descriptive naming—for example, U+0743 SYRIAC TWO VERTICAL DOTS ABOVE.

Syriac Abbreviation Mark. U+070F SYRIAC ABBREVIATION MARK (SAM) is a zero-width formatting code that has no effect on the shaping process of Syriac characters. The SAM specifies the beginning point of a *Syriac abbreviation*, which is a line drawn horizontally above one or more characters, at the end of a word or of a group of characters followed by a character other than a Syriac letter or diacritical mark. A Syriac abbreviation may contain Syriac diacritics.

Ideally, the Syriac abbreviation is rendered by a line that has a dot at each end and the center, as shown in the examples. While not preferable, it has become acceptable for computers to render the Syriac abbreviation as a line without the dots. The line is

acceptable for the presentation of Syriac in plain text, but the presence of dots is recommended in liturgical texts.

The Syriac abbreviation is used for letter numbers and contractions. A Syriac abbreviation generally extends from the last tall character in the word until the end of the word. A common exception to this rule is found with letter numbers that are preceded by a preposition character, as seen in *Figure 9-9*.

Figure 9-9. Syriac Abbreviation

ܝܗ   = 15 (number in letters)

ܒܝܗ   = on the 15th (number with prefix)

ܠܝܣܘܥ   =   ܝܣܘܥܠ

ܩܠܝ   =   ܐܕܡܝܠܩ
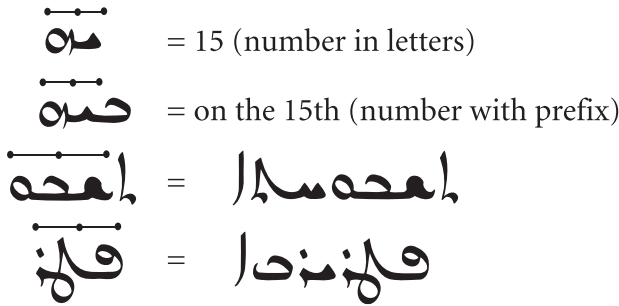
A SAM is placed before the character where the abbreviation begins. The Syriac abbreviation begins over the character following the SAM and continues until the end of the word. Use of the SAM is demonstrated in *Figure 9-10*.

Figure 9-10. Use of SAM

Memory representation: ܟܝ SAM ܗܘ

After reordering: ܗܘ ܝ SAM ܟ

After joining: ܟܝܗܘ
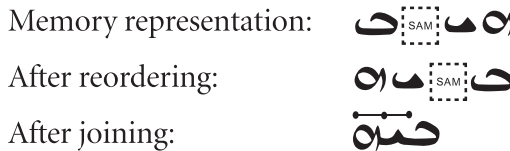
Note: Modern East Syriac texts employ a punctuation mark for contractions of this sort.

Ligatures and Combining Characters. Only one ligature is included in the Syriac block: U+071E SYRIAC LETTER YUDH HE. This combination is used as a unique character in the same manner as an "æ" ligature. A number of combining diacritics unique to Syriac are encoded, but combining characters from other blocks are also used, especially from the Arabic block.

Diacritical Marks and Vowels. The function of the diacritical marks varies. They indicate vowels (as in Arabic and Hebrew), mark grammatical attributes (for example, verb versus noun, interjection), or guide the reader in the pronunciation and/or reading of the given text.

"The reader of the average Syriac manuscript or book is confronted with a bewildering profusion of points. They are large, of medium size and small, arranged singly or in twos and threes, placed above the word, below it, or upon the line."

There are two vocalization systems. The first, attributed to Jacob of Edessa (633–708 CE), utilizes letters derived from Greek that are placed above (or below) the characters they modify. The second is the more ancient dotted system, which employs dots in various shapes and locations to indicate vowels. East Syriac texts exclusively employ the dotted system, whereas West Syriac texts (especially later ones and in modern times) employ a mixture of the two systems.

Diacritical marks are nonspacing and are normally centered above or below the character. Exceptions to this rule follow:

1. U+0741 SYRIAC QUSHSHAYA and U+0742 SYRIAC RUKKAKHA are used only with the letters *beth*, *gamal* (in its Syriac and Garshuni forms), *dalath*, *kaph*, *pe,* and *taw*.

   ○ The *qushshaya* indicates that the letter is pronounced hard and unaspirated.

   ○ The *rukkakha* indicates that the letter is pronounced soft and aspirated. When the *rukkakha* is used in conjunction with the *dalath*, it is printed slightly to the right of the *dalath*'s dot below.

2. In Modern Syriac usage, when a word contains a *rish* and a *seyame*, the dot of the *rish* and the *seyame* are replaced by a *rish* with two dots above it.

3. The *feminine dot* is usually placed to the left of a final *taw*.

Punctuation. Most punctuation marks used with Syriac are found in the Latin-1 and Arabic blocks. The other marks are encoded in this block.

Digits. Modern Syriac employs European numerals, as does Hebrew. The ordering of digits follows the same scheme as in Hebrew.

Harklean Marks. The Harklean marks are used in the Harklean translation of the New Testament. U+070B SYRIAC HARKLEAN OBELUS and U+070D SYRIAC HARKLEAN ASTERISCUS mark the beginning of a phrase, word, or morpheme that has a marginal note. U+070C SYRIAC HARKLEAN METOBELUS marks the end of such sections.

Dalath and Rish. Prior to the development of pointing, early Syriac texts did not distinguish between a *dalath* and a *rish*, which are distinguished in later periods with a dot below the former and a dot above the latter. Unicode provides U+0716 SYRIAC LETTER DOTLESS DALATH RISH as an ambiguous character.

Semkath. Unlike other letters, the joining mechanism of *semkath* varies through the course of history from right-joining to dual-joining. It is necessary to enter a U+200C ZERO WIDTH NON-JOINER character after the *semkath* to obtain the right-joining form where required. Two common variants of this character exist: U+0723 SYRIAC LETTER SEMKATH and U+0724 SYRIAC LETTER FINAL SEMKATH. They occur interchangeably in the same document, similar to the case of Greek sigma.

Vowel Marks. The so-called Greek vowels may be used above or below letters. As West Syriac texts employ a mixture of the Greek and dotted systems, both versions are accounted for here.

Miscellaneous Diacritics. Miscellaneous general diacritics are used in Syriac text. Their usage is explained in *Table 9-14*.

Table 9-14. Miscellaneous Syriac Diacritic Use

Code Points	Use
U+0303, U+0330	These are used in Swadaya to indicate letters not found in Syriac.
U+0304, U+0320	These are used for various purposes ranging from phonological to grammatical to orthographic markers.
U+0307, U+0323, U+1DF8, U+1DFA	These points are used for various purposes—grammatical, phonological, and otherwise. They differ typographically and semantically from the *qushshaya*, *rukkakha* points, and the dotted vowel points. If the point appears above or below a single letter, U+0307 or U+0323 should be used. In contrast, if the point appears between two letters (above or below), U+1DF8 or U+1DFA should be used following the first letter in the encoded character sequence.
U+0308	This is the plural marker. It is also used in Garshuni for the Arabic *teh marbuta*.
U+030A, U+0325	These are two other forms for the indication of *qushshaya* and *rukkakha*. They are used interchangeably with U+0741 SYRIAC QUSHSHAYA and U+0742 SYRIAC RUKKAKHA, especially in West Syriac grammar books.
U+0324	This diacritical mark is found in ancient manuscripts. It has a grammatical and phonological function.
U+032D	This is one of the *digit markers*.
U+032E	This is a mark used in late and modern East Syriac texts as well as in Swadaya to indicate a fricative *pe*.

Use of Characters of the Arabic Block. Syriac makes use of several characters from the Arabic block, including U+0640 ARABIC TATWEEL. Modern texts use U+060C ARABIC COMMA, U+061B ARABIC SEMICOLON, and U+061F ARABIC QUESTION MARK. The *shadda* (U+0651) is also used in the core part of literary Syriac on top of a *waw* in the word "O". Arabic *harakat* are used in Garshuni to indicate the corresponding Arabic vowels and diacritics.

9.3.2 Syriac Shaping

Minimum Rendering Requirements. Rendering requirements for Syriac are similar to those for Arabic. The remainder of this section specifies a minimum set of rules that provides legible Syriac joining and ligature substitution behavior.

Joining Types. Each Syriac letter must be depicted by one of a number of possible contextual glyph forms. The appropriate form is determined on the basis of the cursive joining behavior of that character as it interacts with the cursive joining behavior of adjacent characters. The basic joining types are identical to those specified for the Arabic script, and are specified in the file ArabicShaping.txt in the Unicode Character Database. However, there are additional contextual rules which govern the shaping of U+0710 SYRIAC LETTER ALAPH in final position. The additional glyph types associated with final *alaph* are listed in <u>Table 9-15</u>.

<div align="center">

Table 9-15. Syriac Final Alaph Glyph Types

</div>

Glyph Type	Description
A_{fj}	Final joining (alaph only)
A_{fn}	Final non-joining *except* following dalath and rish (alaph only)
A_{fx}	Final non-joining following dalath and rish (alaph only)

In the following rules, *alaph* refers to U+0710 SYRIAC LETTER ALAPH, which has Joining_Group = Alaph.

These rules are intended to augment joining rules for Syriac which would otherwise parallel the joining rules specified for Arabic in <u>Section 9.2, Arabic</u>. Characters with Joining_Type = Transparent are skipped over when applying the Syriac rules for shaping of *alaph*. In other words, the Syriac parallel for Arabic joining rule R1 would take precedence over the *alaph* joining rules.

S1   **An alaph that has a left-joining character to its right and a non-joining character (or end of text) to its left will take the form of A_{fj}.**

$$\{ + \text{ꝏ} \rightarrow | + \text{ꝏ} \rightarrow \text{ꝏ}$$

S2   **An alaph that has a non-left-joining character to its right, except for a character with Joining_Group = Dalath_Rish, and a non-joining character (or end of text) to its left will take the form of A_{fn}.**

$$\{ + \text{ꝏ} \rightarrow | + \text{ꝏ} \rightarrow | \text{ꝏ}$$

S3 An alaph that has a character with Joining_Group = Dalath_Rish to its right and a non-joining character (or end of text) to its left will take the form of A_{fx}.

$$ 2 + \mathbf{\dot{\jmath}} \rightarrow 2 + \mathbf{\dot{\jmath}} \rightarrow \mathbf{\dot{2}} $$

The example in rule S3 is shown in the East Syriac font style.

Malayalam LLA. U+0868 SYRIAC LETTER MALAYALAM LLA normally connects to the right, but because it joins on both sides in some manuscripts, it is designated dual-joining. To represent right-joining *lla*, the ZWNJ should be employed to make sure it does not connect to the left-side letter.

Syriac Character Joining Groups. Syriac characters can be subdivided into shaping groups, based on the behavior of their letter skeletons when shaped in context. The Unicode character property that specifies these groups is called Joining_Group, and is specified in ArabicShaping.txt in the Unicode Character Database. It is described in the subsection on character joining groups in *Section 9.2, Arabic*.

Table 9-16 exemplifies dual-joining Syriac characters and illustrates the forms taken by the letter skeletons in context. This table and the subsequent table use the Serto (West Syriac) font style, whereas the Unicode code charts are in the Estrangela font style.

Table 9-16. Dual-Joining Syriac Characters

Joining Group	X_n	X_r	X_m	X_l	Notes
BETH	ܒ	ܒ	ܒ	ܒ	Includes PERSIAN BHETH
GAMAL	ܓ	ܓ	ܓ	ܓ	Includes GAMAL GARSHUNI and PERSIAN GHAMAL
HETH	ܚ	ܚ	ܚ	ܚ	
TETH	ܛ	ܛ	ܛ	ܛ	Includes TETH GARSHUNI
YUDH	ܝ	ܝ	ܝ	ܝ	
KAPH	ܟ	ܟ	ܟ	ܟ	
KHAPH	ܟ݂	ܟ݂	ܟ݂	ܟ݂	Sogdian
LAMADH	ܠ	ܠ	ܠ	ܠ	
MIM	ܡ	ܡ	ܡ	ܡ	
NUN	ܢ	ܢ	ܢ	ܢ	
SEMKATH	ܣ	ܣ	ܣ	ܣ	
FINAL_SEMKATH	ܤ	ܤ	ܤ	ܤ	
E	ܥ	ܥ	ܥ	ܥ	
PE	ܦ	ܦ	ܦ	ܦ	
REVERSED_PE	ܱ	ܱ	ܱ	ܱ	

FE	ܬ	ܬ	ܬ	ܬ	Sogdian
QAPH	ܩ	ܩ	ܩ	ܩ	
SHIN	ܫ	ܫ	ܫ	ܫ	
MALAYALAM_NGA	ܓ	ܓ	ܓ	ܓ	Suriyani Malayalam
MALAYALAM_NYA	ܓ	ܓ	ܓ	ܓ	
MALAYALAM_TTA	ܬ	ܬ	ܬ	ܬ	
MALAYALAM_NNA	ܓ	ܓ	ܓ	ܓ	
MALAYALAM_NNNA	ܓ	ܓ	ܓ	ܓ	
MALAYALAM_LLA	ܓ	ܓ	ܓ	ܓ	

The skeleton patterns shown in _Table 9-16_ include six of the Garshuni characters encoded in the Syriac Supplement block (U+0860, U+0862..U+0865, U+0868) that are also dual-joining, and have their own joining group values. U+0868 SYRIAC LETTER MALAYALAM LLA, in particular, normally connects only to the right, but occasionally occurs connected on both sides. That letter is given the dual-joining property value. For instances when a right-joining _lla_ occurs in a manuscript, it may be represented with the sequence <0868, ZWNJ>.

Table 9-17 exemplifies right-joining Syriac characters, illustrating the forms they take in context. Right-joining characters have only two distinct forms, for isolated and final contexts, respectively.

Table 9-17. Right-Joining Syriac Characters

Joining Group	X_n	X_r	Notes
DALATH_RISH	ܪ	ܪ	Includes RISH, DOTLESS DALATH RISH, and PERSIAN DHALATH
HE	ܗ	ܗ	
SYRIAC_WAW	ܘ	ܘ	
ZAIN	ܙ	ܙ	
ZHAIN	ܝ	ܝ	Sogdian
YUDH_HE	ܝܗ	ܝܗ	
SADHE	ܨ	ܨ	
TAW	ܬ	ܬ	
MALAYALAM_RA	ܪ	ܪ	Suriyani Malayalam
MALAYALAM_LLLA	ܠ	ܠ	
MALAYALAM_SSA	ܨ	ܨ	

Table 9-17 includes three of the Garshuni characters encoded in the Syriac Supplement block (U+0867, U+0869, U+086A) that are also right-joining, and have their own joining group values. The two other characters encoded in that block, U+0861 SYRIAC LETTER MALAYALAM JA and U+0866 SYRIAC LETTER MALAYALAM BHA, not shown in the tables above, do not connect either to the right or the left.

U+0710 SYRIAC LETTER ALAPH has the Joining_Group = Alaph and is a right-joining character. However, as specified above in rules S1, S2, and S3, its glyph is subject to additional contextual shaping. *Table 9-18* illustrates all of the glyph forms for *alaph* in each of the three major Syriac type styles.

Table 9-18. Syriac Alaph Glyph Forms

Type Style	X_n	X_r	A_{fj}	A_{fn}	A_{fx}
Estrangela	ܐ	ܐ	ܐ	ܐ	ܐ
Serto (West Syriac)	ܐ	ܐ	ܐ	ܐ	ܐ
East Syriac	ܐ	ܐ	ܐ	ܐ	ܐ

Ligature Classes. As in other scripts, ligatures in Syriac vary depending on the font style. *Table 9-19* identifies the principal valid ligatures for each font style. In some cases, the ligatures are obligatory; those cases are highlighted in bold italic in the table.

Table 9-19. Syriac Ligatures

Characters	Estrangela	Serto (West Syriac)	East Syriac	Sources
ALAPH LAMADH	N/A	***Dual-joining***	N/A	Beth Gazo
GAMAL LAMADH	N/A	Dual-joining	N/A	Armalah
GAMAL E	N/A	Dual-joining	N/A	Armalah
HE YUDH	N/A	N/A	Right-joining	Qdom
YUDH TAW	N/A	Right-joining	N/A	Armalah
KAPH LAMADH	N/A	Dual-joining	N/A	Shhimo
KAPH TAW	N/A	Right-joining	N/A	Armalah
LAMADH SPACE ALAPH	N/A	Right-joining	N/A	Nomocanon
LAMADH ALAPH	Right-joining	***Right-joining***	Right-joining	BFBS
LAMADH LAMADH	N/A	Dual-joining	N/A	Shhimo
NUN ALAPH	N/A	Right-joining	N/A	Shhimo
SEMAKATH TETH	N/A	Dual-joining	N/A	Qurobo
SADHE NUN	Right-joining	Right-joining	Right-joining	Mushhotho
RISH SEYAME	***Right-joining***	***Right-joining***	***Right-joining***	BFBS
TAW ALAPH	Right-joining	N/A	Right-joining	Qdom
TAW YUDH	N/A	N/A	Right-joining	

9.3.3 Syriac Supplement: U+0860–U+086F

The Syriac Supplement block contains characters used to write a dialect of Malayalam called Suriyani Malayalam, which is also known as Garshuni (Karshoni) or Syriac Malayalam.

9.4 Samaritan

9.4.1 Samaritan: U+0800–U+083F

The Samaritan script is used today by small Samaritan communities in Israel and the Palestinian Territories to write the Samaritan Hebrew and Samaritan Aramaic languages, primarily for religious purposes. The Samaritan religion is related to an early form of Judaism, but the Samaritans did not leave Palestine during the Babylonian exile, so the script evolved from the linear Old Hebrew script, most likely directly descended from Phoenician (see _Section 10.3, Phoenician_). In contrast, the more recent square Hebrew script associated with Judaism derives from the Imperial Aramaic script (see _Section 10.4, Imperial Aramaic_) used widely in the region during and after the Babylonian exile, and thus well-known to educated Hebrew speakers of that time.

Like the Phoenician and Hebrew scripts, Samaritan has 22 consonant letters. The consonant letters do not form ligatures, nor do they have explicit final forms as some Hebrew consonants do.

Directionality. The Samaritan script is written from right to left. Conformant implementations of Samaritan script must use the Unicode Bidirectional Algorithm. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Vowel Signs. Vowel signs are optional in Samaritan, just as points are optional in Hebrew. Combining marks are used for vowels that follow a consonant, and are rendered above and to the left of the base consonant. With the exception of *o* and *short a*, vowels may have up to three lengths (normal, long, and overlong), which are distinguished by the size of the corresponding vowel sign. *Sukun* is centered above the corresponding base consonant and indicates that no vowel follows the consonant.

Two vowels, *i* and *short a*, may occur in a word-initial position preceding any consonant. In this case, the separate spacing versions U+0828 SAMARITAN MODIFIER LETTER I and U+0824 SAMARITAN MODIFIER LETTER SHORT A should be used instead of the normal combining marks.

When U+0824 SAMARITAN MODIFIER LETTER SHORT A follows a letter used numerically, it indicates thousands, similar to the use of U+05F3 HEBREW PUNCTUATION GERESH for the same purpose in Hebrew.

Consonant Modifiers. The two marks, U+0816 SAMARITAN MARK IN and U+0817 SAMARITAN MARK IN-ALAF, are used to indicate a pharyngeal voiced fricative /ʕ/. These occur immediately following their base consonant and preceding any vowel signs, and are rendered above and to the right of the base consonant.

U+0818 SAMARITAN MARK OCCLUSION "strengthens" the consonant, for example changing /w/ to /b/. U+0819 SAMARITAN MARK DAGESH indicates consonant gemination. The *occlusion* and *dagesh* marks may both be applied to the same consonant, in which case the *occlusion* mark should precede the *dagesh* in logical order, and the *dagesh* is rendered above the *occlusion* mark. The *occlusion* mark is also used to designate personal names to distinguish them from homographs.

Epenthetic yut represents a kind of glide-vowel which interacts with another vowel. It was originally used only with the consonants *alaf*, *iy*, *it*, and *in*, in combination with a vowel sign. The combining U+081B SAMARITAN MARK EPENTHETIC YUT should be used for this purpose. When *epenthetic yut* is not fixed to one of the four consonants listed above, a new behavior evolved in which the mark for the *epenthetic yut* behaves as a spacing character, capable of bearing its own diacritical mark. U+081A SAMARITAN MODIFIER LETTER EPENTHETIC YUT should be used instead to represent the *epenthetic yut* in this context.

Punctuation. Samaritan uses a large number of punctuation characters. U+0830 SAMARITAN PUNCTUATION NEQUDAA and U+0831 SAMARITAN PUNCTUATION AFSAAQ ("interruption") are similar to the Hebrew *sof pasuq* and were originally used to separate sentences, and later to mark lesser breaks within a sentence. They have also been described respectively as "semicolon" and "pause." Samaritan also uses a smaller dot as a word separator, which can be represented by U+2E31 WORD SEPARATOR MIDDLE DOT. U+083D SAMARITAN PUNCTUATION SOF MASHFAAT is equivalent to the full stop. U+0832 SAMARITAN PUNCTUATION ANGED ("restraint") indicates a break somewhat less strong than an *afsaaq*. U+083E SAMARITAN PUNCTUATION ANNAAU ("rest") is stronger than the *afsaaq* and indicates that a longer time has passed between actions narrated in the sentences it separates.

U+0839 SAMARITAN PUNCTUATION QITSA is similar to the *annaau* but is used more frequently. The *qitsa* marks the end of a section, and may be followed by a blank line to further make the point. It has many glyph variants. One important variant, U+0837 SAMARITAN PUNCTUATION MELODIC QITSA, differs significantly from any of the others, and indicates the end of a sentence "which one should read melodically."

Many of the punctuation characters are used in combination with each other, for example: *afsaaq* + *nequdaa* or *nequdaa* + *afsaaq*, *qitsa* + *nequdaa*, and so on.

U+0836 SAMARITAN ABBREVIATION MARK follows an abbreviation. U+082D SAMARITAN MARK NEQUDAA is an editorial mark which indicates that there is a variant reading of the word.

Other Samaritan punctuation characters mark some prosodic or performative attributes of the text preceding them, as summarized in *Table 9-20*.

Table 9-20. Samaritan Performative Punctuation Marks

Code Point	Name	Description
0833	*bau*	request, prayer, humble petition
0834	*atmaau*	expression of surprise
0835	*shiyyaalaa*	question
0838	*ziqaa*	shout, cry
083A	*zaef*	outburst indicating vehemence or anger
083B	*turu*	didactic expression, a "teaching"
083C	*arkaanu*	expression of submissiveness

9.5 Mandaic

9.5.1 Mandaic: U+0840–U+085F

The origins of the Mandaic script are unclear, but it is thought to have evolved between the second and seventh century CE from a cursivized form of the Aramaic script (as did the Syriac script) or from the Parthian chancery script. It was developed by adherents of the Mandaean gnostic religion of southern Mesopotamia to write the dialect of Eastern Aramaic they used for liturgical purposes, which is referred to as Classical Mandaic.

The religion has survived into modern times, with more than 50,000 Mandaeans in several communities worldwide (most having left what is now Iraq). In addition to the Classical Mandaic still used within some of these communities, a variety known as Neo-Mandaic or Modern Mandaic has developed and is spoken by a small number of people. Mandaeans consider their script sacred, with each letter having specific mystic properties, and the script has changed very little over time.

Letter It. The character U+0847 ‎ MANDAIC LETTER IT is a pharyngeal, pronounced [hu]. It can appear at the end of personal names or at the end of words to indicate the third person singular suffix.

Structure. Mandaic is unusual among Semitic scripts in being a true alphabet; the letters *halqa*, *ushenna*, *aksa*, and *in* are used to write both long and short forms of vowels, instead of functioning as consonants also used to write long vowels (*matres lectionis*), in the manner characteristic of other Semitic scripts. This is possible because some consonant sounds represented by the corresponding letters in other Semitic scripts are not used in the Mandaic language.

The character U+0856 MANDAIC LETTER DUSHENNA, also called *adu*, has a morphemic function. It is used to write the relative pronoun and the genitive exponent *di*. *Dushenna* is a digraph derived from an old ligature for *ad* + *aksa*. It is thus an addition to the usual Semitic set of 22 characters. The Mandaic alphabet is traditionally represented as the 23 letters *halqa* through *dushenna*, with *halqa* appended again at the end to form a symbolically-important cycle of 24 letters.

Two additional Mandaic characters are encoded in the Unicode Standard: U+0858 MANDAIC LETTER AIN is a borrowing from U+0639 ARABIC LETTER AIN. The second additional character, U+0857 MANDAIC LETTER KAD, is a digraph used to write the word *kd*, which means "when, as, like". There are two ways to represent *kad* in Mandaic: U+0857 MANDAIC LETTER KAD or the sequence <U+084A MANDAIC LETTER AK, U+0856 MANDAIC LETTER DUSHENNA>.

The Joining_Type values for U+0856 MANDAIC LETTER DUSHENNA, U+0857 MANDAIC LETTER KAD, and U+0858 MANDAIC LETTER AIN were changed in Unicode Version 13.0 from Non_Joining to Right_Joining. See *Table 9-22*. In cases where the isolated form of *dushenna*, *ain*, or *kad* following a right join-causing character is desired, a U+200C ZERO WIDTH NON-JOINER should be employed to prevent joining with the previous character. (See *Table 9-4* for the definition of a right join-causing character.)

Three diacritical marks are used in teaching materials to differentiate vowel quality; they may be omitted from ordinary text. U+0859 MANDAIC AFFRICATION MARK is used to extend the character set for foreign sounds (whether affrication, lenition, or another sound). U+085A MANDAIC VOCALIZATION MARK is used to distinguish vowel quality of *halqa*, *ushenna*, and *aksa*. U+085B MANDAIC GEMINATION MARK is used to indicate what native writers call a "hard" pronunciation.

Punctuation. Sentence punctuation is used sparsely. A single script-specific punctuation mark is encoded: U+085E MANDAIC PUNCTUATION. It is used to start and end text sections, and is also used in colophons—the historical lay text added to the religious text—where it is typically displayed in a smaller size.

Directionality. The Mandaic script is written from right to left. Conformant implementations of Mandaic script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Shaping and Layout Behavior. Mandaic has fully-developed joining behavior, with forms as shown in *Table 9-21* and *Table 9-22*. In these tables, X_n, X_r, X_m, and X_l designate the nominal, right-joining, dual-joining (medial), and left-joining forms respectively, just as in *Table 9-6*, *Table 9-7*, and *Table 9-8*.

Table 9-21. Dual-Joining Mandaic Characters

Character	X_n	X_r	X_m	X_l
AB	ﻌ	ﻌ	ﻌ	ﻌ
AG	ﻋ	ﻋ	ﻋ	ﻋ
AD	ﺪ	ﺪ	ﺪ	ﺪ
AH	ﻤ	ﻤ	ﻤ	ﻤ
USHENNA	ﺪ	ﺪ	ﺪ	ﺪ
ATT	ﻟ	ﻟ	ﻟ	ﻟ
AK	ﻴ	ﻴ	ﻴ	ﻴ
AL	ﺎ	ﺎ	ﺎ	ﺎ
AM	ﺊ	ﺊ	ﺊ	ﺊ
AN	ﻴ	ﻴ	ﻴ	ﻴ
AS	�	ﺺ	ﺺ	ﺺ

IN	ڪ	ڪ	ڪ	ڪ
AP	‏ﯾ	‏ﯾ	‏ﭘ	‏ﭘ
ASZ	‏ﺳﻳ	‏ﺳﻳ	‏ﺷﯾ	‏ﺷﯾ
AQ	ﻪﻛ	ﻪﻛ	ﻪﻛ	ﻪﻛ
AR	ﻪﻛ	ﻪﻛ	ﻪﻛ	ﻪﻛ
AT	ﺗﺤ	ﺗﺤ	ﺗﺤ	ﺗﺤ

Table 9-22. Right-Joining Mandaic Characters

Character	X_n	X_r
HALQA	٥	٥
AZ	ا	ا
IT	٥	٥
AKSA	ﺣ	ﺣ
ASH	٩٥	٩٥
DUSHENNA	ﻟﻟ	ﻟﻟ
KAD	ﻓﻮ	ﻓﻮ
AIN	ﻉ	ﻉ

Line Breaking. Spaces provide the primary line break opportunity. When text is fully justified, words may be stretched as in Arabic. U+0640 ARABIC TATWEEL may be inserted for this purpose.

9.6 Yezidi

9.6.1 Yezidi: U+10E80–U+10EBF

The Yezidi script was used to write two religious texts, *Masḥafā Reš* and *Ketēbā Jelwa*, which may date to the 12th or 13th centuries. The history of the script between the creation of these texts and the current period is unclear; however, the Spiritual Council of Yezidis in Georgia decided to revive the script in 2013. As part of the revitalization, two specialists modified the script to represent the Yezidi language (called *Êzdîkî* in the vernacular), which is also referred to as the Kurmanji language. This language can also be written in the Latin, Cyrillic, and Arabic scripts. Today, clergy in the Yezidi temple in Tbilisi use the Yezidi script to write prayers, sacred books, and in other contexts.

Structure. Yezidi is an alphabet, written right to left. Ligatures occur in the historical texts, but not in the modern version of the script.

Letters. A set of ten letters have been added to the repertoire to represent the modern Kurmanji language. Two historic letters with diacritics are separately encoded as atomic characters: U+10EB0 YEZIDI LETTER LAM WITH DOT ABOVE and U+10EB1 YEZIDI LETTER YOT WITH CIRCUMFLEX ABOVE. The letters with diacritic marks have distinct pronunciation: YEZIDI LETTER LAM WITH DOT ABOVE is pronounced [ɫ], instead of [l], and YEZIDI LETTER YOT WITH CIRCUMFLEX ABOVE is pronounced [e], instead of [j].

Long *u* is indicated by a ligature of <U+10EA3 YEZIDI LETTER UM, U+10EA3 YEZIDI LETTER UM>. This sequence of two *um* characters may appear kerned or unkerned, without difference in meaning.

Diacritics. Two combining diacritics, U+10EAB YEZIDI COMBINING HAMZA MARK and U+10EAC YEZIDI COMBINING MADDA MARK, appear in words of Arabic origin. Additional diacritics appear in the *Masḥafā Re*š, but the meaning of the marks is unclear, so they are not currently encoded.

Punctuation. U+10EAD YEZIDI HYPHENATION MARK may appear above the last letter in a line to indicate a word break. In historic texts, the hyphenation mark may appear at the beginning of a line or above the last letter in a line. Occasionally, the mark can be used to denote long phonemes within a word, but this usage does not apply to modern texts.

Yezidi also uses U+060C ARABIC COMMA, U+061B ARABIC SEMICOLON, and U+061F ARABIC QUESTION MARK, in addition to U+002E FULL STOP and U+003A COLON.

Numbers. Older texts employ Arabic-Indic numbers (U+0660..U+0669), but Western digits are preferred in modern usage.

Chapter 10

Middle East-II

Ancient Scripts

This chapter covers a number of ancient scripts of the Middle East. All of these scripts were written right to left.

Old North Arabian and Old South Arabian are two branches of the South Semitic script family used in and around Arabia from about the tenth century BCE to the sixth century CE. The Old South Arabian script was used around the southwestern part of the Arabian peninsula for 1,200 years beginning around the 8th century BCE. Carried westward, it was adapted for writing the Ge'ez language, and evolved into the root of the modern Ethiopic script.

The Phoenician alphabet was used in various forms around the Mediterranean. It is ancestral to Latin, Greek, Hebrew, and many other scripts—both modern and historical.

The Imperial Aramaic script evolved from Phoenician and was the source of many other scripts, such as the square Hebrew and the Arabic script. Imperial Aramaic was used to write the Aramaic language beginning in the eighth century BCE, and was the principal administrative language of the Assyrian empire and then the official language of the Achaemenid Persian empire. Inscriptional Parthian, Inscriptional Pahlavi, and Avestan are also derived from Imperial Aramaic, and were used to write various Middle Persian languages.

Psalter Pahlavi is a cursive alphabetic script used to write the Middle Persian language during the 6th or 7th century CE. It is a historically conservative variety of Pahlavi used by Christians in the Neo-Persian empire.

The Chorasmian script was used between the 2nd century and 8th to 9th centuries CE primarily to write the Chorasmian language, an Eastern Iranian language. The script was derived from Imperial Aramaic and is related to Parthian, Inscriptional Pahlavi, Psalter Pahlavi, Book Pahlavi, and Old Sogdian.

The Manichaean script is a cursive alphabetic script related to Syriac, as well as Palmyrene Aramaic. The script was used by those practicing the Manichaean religion, which was founded during the third century CE in Babylonia, and spread widely over the next four centuries before later vanishing.

The Elymaic script was used to write Achaemenid Aramaic in the state of Elymais, which flourished from the second century BCE to the early third century CE and was located in the southwestern portion of modern-day Iran. Elymaic derives from the Aramaic script and is closely related to Parthian and Mandaic.

The Nabataean script developed from the Aramaic script and was used to write the language of the Nabataean kingdom. The script was in wide use from the second century BCE to the fourth century CE. It is generally considered the precursor of the Arabic script.

The Palmyrene script was derived from the customary forms of Aramaic developed during the Achaemenid empire. The script was used for writing the Palmyrene dialect of West Aramaic, and is known from inscriptions and documents found mainly in the city of Palmyra and other cities in the region of Syria, dating from 44 BCE to about 280 CE.

The Hatran script belongs to the North Mesopotamian branch of the Aramaic scripts, and was used for writing a dialect of the Aramaic language. The script is known from inscriptions discovered in the ancient city of Hatra, in present-day Iraq, dating from 98–97 BCE until circa 241 CE.

10.1 Old North Arabian

10.1.1 Old North Arabian: U+10A80–U+10A9F

Old North Arabian, or Ancient North Arabian, refers to a group of scripts used in the western two-thirds of Arabia and the Levant, from Syria to the borders of Yemen. Old North Arabian is a member of the South Semitic script family, which was used exclusively in Arabia and environs, and is a relative of the Old South Arabian script. The earliest datable Old North Arabian texts are from the mid-sixth century BCE. The script is thought to have fallen out of use after the fourth century CE. The encoding of Old North Arabian is based on the Dadanitic form, which is attested in many formal inscriptions on stelae and rock-faces, and hundreds of graffiti used in the oasis of Dadan (Dedān, modern al-'Ulā) in northwest Saudi Arabia.

Other forms of the Old North Arabian script, such as Minaic, Safaitic, Hismaic, Taymanitic and Thamudic B, have many variant forms of the letters. Dialect-specific fonts can be used to render these variant forms.

Structure. Old North Arabian is an alphabetic script consisting only of consonants; vowels are not indicated in the script, though some Dadanitic texts do make limited use of consonant letters to write long vowels (*matres lectionis*). The script has been encoded with right-to-left directionality, which is typical for Dadanitic. Glyphs may be mirrored in lines when they have left-to-right directionality.

Ordering. Traditional sorting orders are poorly attested. Modern scholars specializing in Old North Arabian prefer the South Semitic alphabetical order shown in the code charts.

Numbers. Three numbers are attested in Old North Arabian: one, ten, and twenty. The numbers have right-to-left directionality.

Punctuation. A vertical word separator is usually used between words in Dadanitic, but this is not widely used in the other Old North Arabian alphabets. U+10A9D OLD NORTH ARABIAN NUMBER ONE is used to represent both this punctuation and the digit one.

10.2 Old South Arabian

10.2.1 Old South Arabian: U+10A60–U+10A7F

The Old South Arabian script was used on the Arabian peninsula (especially in what is now Yemen) from the 8th century BCE to the 6th century CE, after which it was supplanted by the Arabic script. It is a consonant-only script of 29 letters, and was used to write the southwest Semitic languages of various cultures: Minean, Sabaean, Qatabanian, Hadramite, and Himyaritic. Old South Arabian is thus known by several other names including Mino-Sabaean, Sabaean and Sabaic. It is attested primarily in an angular form ("Musnad") in monumental inscriptions on stone, ceramic material, and metallic surfaces; however, since the mid 1970s examples of a more cursive form ("Zabur") have been found on softer materials, such as wood and leather.

Around the end of the first millennium BCE, the westward migration of the Sabaean people into the Horn of Africa introduced the South Arabic script into the region, where it was adapted for writing the Ge'ez language. By the 4th century CE the script for Ge'ez had begun to change, and eventually evolved into a left-to-right syllabary with full vowel representation, the root of the modern Ethiopic script (see *Section 19.1, Ethiopic*).

Directionality. The Old South Arabian script is typically written from right to left. Conformant implementations of Old South Arabian script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm"). However, some older examples of the script are written in boustrophedon style, with glyphs mirrored in lines with left-to-right directionality.

Structure. The character repertoire of Old South Arabian corresponds to the repertoire of Classical Arabic, plus an additional letter presumed analogous to the letter *samekh* in West Semitic alphabets. This results in four letters for different kinds of "s'"sounds. While there is no general system for representing vowels, the letters U+10A65 OLD SOUTH ARABIAN LETTER WAW and U+10A7A OLD SOUTH ARABIAN LETTER YODH can also be used to represent the long vowels *u* and *i*. There is no evidence of any kind of diacritical marks; geminate consonants are indicated simply by writing the corresponding letter twice, for example.

Segmentation. Letters are written separately, there are no connected forms. Words are not separated with space; word boundaries are instead marked with a vertical bar. The vertical bar is indistinguishable from U+10A7D "ꝯ" OLD SOUTH ARABIAN NUMBER ONE—only one character is encoded to serve both functions. Words are broken arbitrarily at line boundaries in attested materials.

Monograms. Several letters are sometimes combined into a single group, in which the glyphs for the constituent characters are overlaid and sometimes rotated to create

what appears to be a single unit. These combined units are traditionally called *monograms* by scholars of this script.

Numbers. Numeric quantities are differentiated from surrounding text by writing U+10A7F 𐩿 OLD SOUTH ARABIAN NUMERIC INDICATOR before and after the number. Six characters have numeric values as shown in *Table 10-1*—four of these are letters that double as numeric values, and two are characters not used as letters.

Table 10-1. Old South Arabian Numeric Characters

Code Point	Glyph	Numeric function	Other function
10A7F	𐩿	numeric separator	
10A7D	ı	1	word separator
10A6D	ᕼ	5	kheth
10A72	о	10	ayn
10A7E	ᕯ	50	
10A63	ⱶ	100	mem
10A71	ħ	1000	alef

Numbers are built up through juxtaposition of these characters in a manner similar to that of Roman numerals, as shown in *Table 10-2*. When 10, 50, or 100 occur preceding 1000 they serve to indicate multiples of 1000. The example numbers shown in *Table 10-2* are rendered in a right-to-left direction in the last column.

Table 10-2. Number Formation in Old South Arabian

Value	Schematic	Character Sequence	Display
1	1	10A7D	ı
2	1 + 1	10A7D 10A7D	ıı
3	1 + 1 + 1	10A7D 10A7D 10A7D	ııı
5	5	10A6D	ᕼ
7	5 + 1 + 1	10A6D 10A7D 10A7D	ııᕼ
16	10 + 5 + 1	10A72 10A6D 10A7D	ıᕼо
1000	1000	10A71	ħ
3000	1000 + 1000 + 1000	10A71 10A71 10A71	ħħħ
10000	10 × 1000	10A72 10A71	ħо
11000	10 × 1000 + 1000	10A72 10A71 10A71	ħħо
30000	(10 + 10 + 10) × 1000	10A72 10A72 10A72 10A71	ħооо
30001	(10 + 10 + 10) × 1000 + 1	10A72 10A72 10A72 10A71 10A7D	ıħооо

Character Names. Character names are based on those of corresponding letters in northwest Semitic.

10.3 Phoenician

10.3.1 Phoenician: U+10900–U+1091F

The Phoenician alphabet and its successors were widely used over a broad area surrounding the Mediterranean Sea. Phoenician evolved over the period from about the twelfth century BCE until the second century BCE, with the last neo-Punic inscriptions dating from about the third century CE. Phoenician came into its own from the ninth century BCE. An older form of the Phoenician alphabet is a fore-runner of the Greek, Old Italic (Etruscan), Latin, Hebrew, Arabic, and Syriac scripts among others, many of which are still in modern use. It has also been suggested that Phoenician is the ultimate source of Kharoshthi and of the Indic scripts descending from Brahmi.

Phoenician is an historic script, and as for many other historic scripts, which often saw continuous change in use over periods of hundreds or thousands of years, its delineation as a script is somewhat problematic. This issue is particularly acute for historic Semitic scripts, which share basically identical repertoires of letters, which are historically related to each other, and which were used to write closely related Semitic languages.

In the Unicode Standard, the Phoenician script is intended for the representation of text in Paleo-Hebrew, Archaic Phoenician, Phoenician, Early Aramaic, Late Phoeni-cian cursive, Phoenician papyri, Siloam Hebrew, Hebrew seals, Ammonite, Moabite, and Punic. The line from Phoenician to Punic is taken to constitute a single contin-uous branch of script evolution, distinct from that of other related but separately encoded Semitic scripts.

The earliest Hebrew language texts were written in the Paleo-Hebrew alphabet, one of the forms of writing considered to be encompassed within the Phoenician script as encoded in the Unicode Standard. The Samaritans who did not go into exile continued to use Paleo-Hebrew forms, eventually developing them into the distinct Samaritan script. (See *Section 9.4, Samaritan*.) The Jews in exile gave up the Paleo-Hebrew alphabet and instead adopted Imperial Aramaic writing, which was a descendant of the Early Aramaic form of the Phoenician script. (See *Section 10.4, Imperial Aramaic*.) Later, they transformed Imperial Aramaic into the "Jewish Aramaic" script now called (Square) Hebrew, separately encoded in the Hebrew block in the Unicode Standard. (See *Section 9.1, Hebrew*.)

Some scholars conceive of the language written in the Paleo-Hebrew form of the Phoenician script as being quintessentially Hebrew and consistently transliterate it into Square Hebrew. In such contexts, Paleo-Hebrew texts are often considered to simply *be* Hebrew, and because the relationship between the Paleo-Hebrew letters and Square Hebrew letters is one-to-one and quite regular, the transliteration is

conceived of as simply a font change. Other scholars of Phoenician transliterate texts into Latin. The encoding of the Phoenician script in the Unicode Standard does not invalidate such scholarly practice; it is simply intended to make it possible to represent Phoenician, Punic, and similar textual materials directly in the historic script, rather than as specialized font displays of transliterations in modern Square Hebrew.

Directionality. Phoenician is written horizontally from right to left. The characters of the Phoenician script are all given strong right-to-left directionality.

Punctuation. Inscriptions and other texts in the various forms of the Phoenician script generally have no space between words. Dots are sometimes found between words in later exemplars—for example, in Moabite inscriptions—and U+1091F PHOENICIAN WORD SEPARATOR should be used to represent this punctuation. The appearance for this word separator is somewhat variable; in some instances it may appear as a short vertical bar, instead of a rounded dot.

Stylistic Variation. The letters for Phoenician proper and especially for Punic have very exaggerated descenders. These descenders help distinguish the main line of Phoenician script evolution toward Punic, as contrasted with the Hebrew forms, where the descenders instead grew shorter over time.

Numerals. Phoenician numerals are built up from six elements used in combination. These include elements for one, two, and three, and then separate elements for ten, twenty, and one hundred. Numerals are constructed essentially as tallies, by repetition of the various elements. The numbers for two and three are graphically composed of multiples of the tally mark for one, but because in practice the values for two or three are clumped together in display as entities separate from one another they are encoded as individual characters. This same structure for numerals can be seen in some other historic scripts ultimately descendant from Phoenician, such as Imperial Aramaic and Inscriptional Parthian.

Like the letters, Phoenician numbers are written from right to left: |||⅋⅋∿ means 143 (100 + 20 + 20 + 3). This practice differs from modern Semitic scripts like Hebrew and Arabic, which use decimal numbers written from left to right.

Character Names. The names used for the characters here are those reconstructed by Theodor Nöldeke in 1904, as given in Powell (1996).

10.4 Imperial Aramaic

10.4.1 Imperial Aramaic: U+10840–U+1085F

The Aramaic language and script are descended from the Phoenician language and script. Aramaic developed as a distinct script by the middle of the eighth century BCE and soon became politically important, because Aramaic became first the principal administrative language of the Assyrian empire, and then the official language of the Achaemenid Persian empire beginning in 549 BCE. The Imperial Aramaic script was the source of many other scripts, including the square Hebrew script, the Arabic script, and scripts used for Middle Persian languages, including Inscriptional Parthian, Inscriptional Pahlavi, and Avestan.

Imperial Aramaic is an alphabetic script of 22 consonant letters but no vowel marks. It is written either in *scriptio continua* or with spaces between words.

Directionality. The Imperial Aramaic script is written from right to left. Conformant implementations of the script must use the Unicode Bidirectional Algorithm. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm".

Punctuation. U+10857 IMPERIAL ARAMAIC SECTION SIGN is thought to be used to mark topic divisions in text.

Numbers. Imperial Aramaic has its own script-specific numeric characters with right-to-left directionality. Numbers are built up using sequences of characters for 1, 2, 3, 10, 20, 100, 1000, and 10000 as shown in *Table 10-3*. The example numbers shown in the last column are rendered in a right-to-left direction.

Table 10-3. Number Formation in Aramaic

Value	Schematic	Character Sequence	Display
1	1	10858	𐡘
2	2	10859	𐡙
3	3	1085A	𐡚
4	3 + 1	1085A 10858	𐡘𐡚
5	3 + 2	1085A 10859	𐡙𐡚
9	3 + 3 + 3	1085A 1085A 1085A	𐡚𐡚𐡚
10	10	1085B	𐡛
11	10 + 1	1085B 10858	𐡘𐡛
12	10 + 2	1085B 10859	𐡙𐡛
20	20	1085C	𐡜
30	20 + 10	1085C 1085B	𐡛𐡜

55	20 + 20 + 10 + 3 + 2	1085C 1085C 1085B 1085A 10859	𐡜𐡛𐡚𐡙𐡙
70	20 + 20 + 20 + 10	1085C 1085C 1085C 1085B	𐡚𐡛𐡛𐡛
100	1 × 100	10858 1085D	𐡝𐡘
200	2 × 100	10859 1085D	𐡝𐡙
500	(3 + 2) × 100	1085A 10859 1085D	𐡝𐡙𐡚
3000	3 × 1000	1085A 1085E	𐡞𐡚
30000	3 × 10000	1085A 1085F	𐡟𐡚

Values in the range 1-99 are represented by a string of characters whose values are in the range 1-20; the numeric value of the string is the sum of the numeric values of the characters. The string is written using the minimum number of characters, with the most significant values first. For example, 55 is represented as 20 + 20 + 10 + 3 + 2. Characters for 100, 1000, and 10000 are prefixed with a multiplier represented by a string whose value is in the range 1-9. The Inscriptional Parthian, Inscriptional Pahlavi, Nabataean, Palmyrene, and Hatran scripts use a similar system for forming numeric values.

10.5 Manichaean

10.5.1 Manichaean: U+10AC0–U+10AFF

The Manichaean religion was founded during the third century CE in Babylonia, then part of the Sassanid Persian empire. It spread widely over the next four centuries, as far west as north Africa and as far east as China, but had mostly vanished by the fourteenth century. From 762 until around 1000 it was a state religion in the Uyghur kingdom.

The Manichaean script was used by adherents of Manichaeism, and was based on or influenced by the Estrangela form of Syriac, as well as Palmyrene Aramaic. It is said to have been invented by Mani, but may be older. Because of the wide spread of Manichaeism and Mani's decision to spread his teachings in any language available, the Manichaean script was used to write a variety of languages with some variation in character repertoire: the Iranian languages Middle and Early Modern Persian, Parthian, Sogdian, and Bactrian, as well as the Turkic language Uyghur and, to a lesser extent, the Indo-European language Tocharian.

Directionality. The Manichaean script is written from right to left. Conformant implementations of Manichaean script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Structure. Manichaean is alphabetic, written with spaces between words. The alphabet includes 24 base letters, two more than Aramaic. There are a total of 36 letters. Ten of these are formed by adding one or two dots above the base letter to represent a spirant or other modified sound. There is also a sign representing the conjunction *ud*.

In addition, two diacritical marks are used to indicate abbreviations, elisions, or plural forms. Manichaean text paid careful attention to the layout of characters, often stretching or shrinking letters, using abbreviations, or eliminating vowels (indicated with elision dots) to achieve desired line widths and to avoid breaking words across lines. Sogdian written in Manichaean script also sometimes shows the use of doubled vowels to fill out a line.

To graphically extend a word, U+0640 ARABIC TATWEEL may be used.

Shaping. Manichaean has shaping rules and rendering requirements that are similar to those for Syriac and Arabic, with joining forms as shown in *Table 10-4*, *Table 10-5*, *Table 10-6* and *Table 10-7*. In these tables, X_n, X_r, X_m, and X_l designate the isolated, final, medial, and initial forms respectively. The dotted letters are not shown separately, because their joining behavior is the same as the corresponding un-dotted

letter. Note that Manichaean has two letters with the rare Joining_Type of Left_Joining.

Five Manichaean letters—*daleth*, *he*, *mem*, *nun*, *resh*—have alternate forms whose occurrence cannot be predicted from context, although the alternate forms tend to occur most often at the end of lines. These forms are represented using standardized variation sequences and are shown in the tables that follow.

Table 10-4 lists the dual-joining letters Manichaean. In this and the following tables, the standardized variation sequences are indicated in the joining group column in separate rows showing the relevant joining group plus the variation selector.

Table 10-4. Dual-Joining Manichaean Letters

Joining Group	X_n	X_r	X_m	X_l
ALEPH				
BETH				
GIMEL				
GHIMEL				
LAMEDH				
DHAMEDH				
THAMEDH				
MEM				
MEM + VS-1				
SAMEKH				
AYIN				
PE				
QOPH				

Table 10-5 lists the right-joining letters for Manichaean.

Table 10-5. Right-Joining Manichaean Letters

Joining Group	X_n	X_r
DALETH		
DALETH + VS-1		
WAW		
ZAYIN		
TETH		
YODH		
KAPH		
SADHE		

RESH	ࡘ	ࡘ
RESH + VS-1	ࡘ	ࡘ
TAW	࡚	࡚

Table 10-6 lists the left-joining letters for Manichaean.

Table 10-6. Left-Joining Manichaean Letters

Joining Group	X_n	X_l
HETH	ࡍ	ࡍ
NUN	ࡌ	ࡌ
NUN + VS-1	ࡌ	ࡌ

Table 10-7 lists the non-joining letters for Manichaean

Table 10-7. Non-Joining Manichaean Letters

Joining Group	X_n
HE	ࡕ
HE + VS-1	ࡕ
JAYIN	ࡘ
SHIN	ࡔ

Manichaean has two obligatory ligatures for *sadhe* followed by *yodh* or *nun*. These are shown in *Table 10-8*.

Table 10-8. Manichaean Ligatures

Character Sequence	X_n	X_r
SADHE + YODH	ࡑ	ࡑ
SADHE + NUN	ࡑ	ࡑ

Numbers. Manichaean has script-specific numeric characters with right-to-left directionality. Numbers are built up using sequences of characters for 1, 5, 10, 20, and 100 in a manner which appears similar to Imperial Aramaic number formation (see *Table 10-3*); however, very few numeric values are attested in Manichaean sources. Manichaean numeric characters exhibit contextual joining behavior, as with letters, but the existing sources do not demonstrate all of the forms.

Punctuation. Manichaean consistently uses a number of script-specific punctuation marks. U+10AF0 MANICHAEAN PUNCTUATION STAR is used to mark the beginning and end of headlines; U+10AF1 MANICHAEAN PUNCTUATION FLEURON and U+10AF5 MANICHAEAN PUNCTUATION TWO DOTS are used to mark the beginning

and end of headlines and captions. U+10AF6 MANICHAEAN PUNCTUATION LINE FILLER is used as a sort of ellipsis to fill out a line.

U+10AF2 MANICHAEAN PUNCTUATION DOUBLE DOT WITHIN DOT is used to indicate larger units of text in a prose text or the end of a strophe in a verse text. U+10AF3 MANICHAEAN PUNCTUATION DOT WITHIN DOT is used to indicate smaller units of text in a prose text or the end of a half-verse in a verse text. U+10AF4 MANICHAEAN PUNCTUATION DOT is used to indicate sub-units of text, logical parts of a sentence or units in a list.

10.6 Pahlavi and Parthian

The Inscriptional Parthian script was used to write Parthian and other languages. It had evolved from the Imperial Aramaic script by the second century CE, and was used as an official script during the first part of the Neo-Persian (Sasanian) empire. It is attested primarily in surviving inscriptions, the last of which dates from 292 CE. Inscriptional Pahlavi also evolved from the Aramaic script during the second century CE during the late period of the Parthian Persian empire in what is now southern Iran. It was used as a monumental script to write Middle Persian until the fifth century CE.

Psalter Pahlavi is a cursive alphabetic script that was used to write the Middle Persian language during the 6th or 7th century CE. It is a historically conservative variety of Pahlavi used by Christians in the Neo-Persian empire. The name of the script is based on its main attestation in a fragmentary manuscript of the Psalms of David, known as the Pahlavi Psalter. The later Book Pahlavi is another variety of the script.

10.6.1 Inscriptional Parthian: U+10B40–U+10B5F

Inscriptional Pahlavi: U+10B60–U+10B7F

Inscriptional Parthian and Inscriptional Pahlavi are both alphabetic scripts and are usually written with spaces between words. Inscriptional Parthian has 22 consonant letters but no vowel marks, while Inscriptional Pahlavi consists of 19 consonant letters; two of which are used for writing multiple consonants, so that it can be used for writing the usual Phoenician-derived 22 consonants.

Directionality. Both the Inscriptional Parthian script and the Inscriptional Pahlavi script are written from right to left. Conformant implementations must use the Unicode Bidirectional Algorithm. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Shaping and Layout Behavior. Inscriptional Parthian makes use of seven standard ligatures. Ligation is common, but not obligatory; U+200C ZERO WIDTH NON-JOINER can be used to prevent ligature formation. The same glyph is used for both the *yodh-waw* and *nun-waw* ligatures. The letters *sadhe* and *nun* have swash tails which typically trail under the following letter; thus two *nun*s will nest, and the tail of a *nun* that precedes a *daleth* may be displayed between the two parts of the *daleth* glyph. *Table 10-9* shows these behaviors.

Table 10-9. Inscriptional Parthian Shaping Behavior

⅂	*(gimel)*	+	ↄ	*(waw)*	→	ↄⅆ	*(gw)*
⅄	*(heth)*	+	ↄ	*(waw)*	→	ↄ⅄	*(xw)*

‍	*(yodh)*	+	‍	*(waw)*	→	‍	*(yw)*
‍	*(nun)*	+	‍	*(waw)*	→	‍	*(nw)*
‍	*(ayin)*	+	‍	*(lamedh)*	→	‍	*('l)*
‍	*(resh)*	+	‍	*(waw)*	→	‍	*(rw)*
‍	*(taw)*	+	‍	*(waw)*	→	‍	*(tw)*
‍	*(nun)*	+	‍	*(nun)*	→	‍	*(nn)*
‍	*(nun)*	+	‍	*(daleth)*	→	‍	*(nd)*

In Inscriptional Pahlavi, U+10B61 INSCRIPTIONAL PAHLAVI LETTER BETH has a swash tail which typically trails under the following letter, similar to the behavior of U+10B4D INSCRIPTIONAL PARTHIAN LETTER NUN.

Numbers. Inscriptional Parthian and Inscriptional Pahlavi each have script-specific numeric characters with right-to-left directionality. Numbers in both are built up using sequences of characters for 1, 2, 3, 4, 10, 20, 100, and 1000 in a manner similar to the way numbers are built up for Imperial Aramaic; see *Table 10-3*. In Inscriptional Parthian the units are sometimes written with strokes of the same height, or sometimes written with a longer ascending or descending final stroke to show the end of the number.

Heterograms. As scripts derived from Aramaic (such as Inscriptional Parthian and Pahlavi) were adapted for writing Iranian languages, certain words continued to be written in the Aramaic language but read using the corresponding Iranian-language word. These are known as heterograms or xenograms, and were formerly called "ideograms".

10.6.2 Psalter Pahlavi: U+10B80–U+10BAF

Structure. Psalter Pahlavi is an alphabetic script written from right to left. It uses spaces between words. The script has fully-developed cursive joining behavior. To graphically extend a word, U+0640 ARABIC TATWEEL may be used.

Numbers. Psalter Pahlavi has its own numbers, which also have right-to-left directionality. Numbers are built up out of 1, 2, 3, 4, 10, 20, and 100. Some Psalter Pahlavi numbers have joining behavior, and can join with letters as well as numbers.

Punctuation. There are four types of large section-ending punctuation. The most common is U+10B99 PSALTER PAHLAVI SECTION MARK, which is written with red dots in the vertical position and black dots in the horizontal position; the red dots are often written as rings. Less common but found together with this is U+10B9A PSALTER PAHLAVI TURNED SECTION MARK, which is written with black dots in the vertical position and red dots in the horizontal position. More rare are U+10B9B PSALTER PAHLAVI FOUR DOTS WITH CROSS (sometimes found immediately following the *section mark*), and U+10B9C PSALTER PAHLAVI FOUR DOTS WITH DOT.

10.7 Avestan

10.7.1 Avestan: U+10B00–U+10B3F

The Avestan script was created around the fifth century CE to record the canon of the Avesta, the principal collection of Zoroastrian religious texts. The Avesta had been transmitted orally in the Avestan language, which was by then extinct except for liturgical purposes. The Avestan script was also used to write the Middle Persian language, which is called Pazand when written in Avestan script. The Avestan script was derived from Book Pahlavi, but provided improved phonetic representation by adding consonants and a complete set of vowels—the latter probably due to the influence of the Greek script. It is an alphabetic script of 54 letters, including one that is used only for Pazand.

Directionality. The Avestan script is written from right to left. Conformant implementations of Avestan script must use the Unicode Bidirectional Algorithm. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm".

Shaping Behavior. Four ligatures are commonly used in manuscripts of the Avesta, as shown in *Table 10-10*. U+200C ZERO WIDTH NON-JOINER can be used to prevent ligature formation.

Table 10-10. Avestan Shaping Behavior

ᵕ	(š)	+	ᴗ	(a)	→	ᵕᴗ	(ša)
ᵕ	(š)	+	ᴘ	(ce)	→	ᴘᵕ	(šc)
ᵕ	(š)	+	ᴑ	(te)	→	ᴑᵕ	(št)
ᴗ	(a)	+	ᴑ	(he)	→	ᴑᴗ	(ah)

Punctuation. Archaic Avestan texts use a dot to separate words. The texts generally use a more complex grouping of dots or other marks to indicate boundaries between larger units such as clauses and sentences, but this is not systematic. In contemporary critical editions of Avestan texts, some scholars have systematized and differentiated the usage of various Avestan punctuation marks. The most notable example is Karl F. Geldner's 1880 edition of the Avesta.

The Unicode Standard encodes a set of Avestan punctuation marks based on the system established by Geldner. U+10B3A TINY TWO DOTS OVER ONE DOT PUNCTUATION functions as an Avestan colon, U+10B3B SMALL TWO DOTS OVER ONE DOT PUNCTUATION as an Avestan semicolon, and U+10B3C LARGE TWO DOTS OVER ONE DOT PUNCTUATION as an Avestan end of sentence mark; these indicate breaks of increasing finality. U+10B3E LARGE TWO RINGS OVER ONE RING PUNCTUATION functions as an Avestan end of section, and may be doubled (sometimes with a space

between) for extra finality. U+10B39 ᴀᴠᴇsᴛᴀɴ ᴀʙʙʀᴇᴠɪᴀᴛɪᴏɴ ᴍᴀʀᴋ is used to mark abbreviation and repetition. U+10B3D ʟᴀʀɢᴇ ᴏɴᴇ ᴅᴏᴛ ᴏᴠᴇʀ ᴛᴡᴏ ᴅᴏᴛs ᴘᴜɴᴄᴛᴜᴀ-ᴛɪᴏɴ and U+10B3F ʟᴀʀɢᴇ ᴏɴᴇ ʀɪɴɢ ᴏᴠᴇʀ ᴛᴡᴏ ʀɪɴɢs ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ are found in Avestan texts, but are not used by Geldner.

Minimal representation of Avestan requires two separators: one to separate words and a second mark used to delimit larger units, such as clauses or sentences. Contemporary editions of Avestan texts show the word separator dot in a variety of vertical positions: it may appear in a midline position or on the baseline. Dots such as U+2E31 ᴡᴏʀᴅ sᴇᴘᴀʀᴀᴛᴏʀ ᴍɪᴅᴅʟᴇ ᴅᴏᴛ, U+00B7 ᴍɪᴅᴅʟᴇ ᴅᴏᴛ, or U+002E ꜰᴜʟʟ sᴛᴏᴘ can be used to represent this.

10.8 Chorasmian

10.8.1 Chorasmian: U+10FB0–U+10FDF

The Chorasmian script was derived from Imperial Aramaic and is related to Parthian, Inscriptional Pahlavi, Psalter Pahlavi, Book Pahlavi, and Old Sogdian. It was used between the 2nd century and the 8th to 9th centuries CE primarily to write the Chorasmian language, a now-extinct Eastern Iranian language. The script and language were used in a region in Central Asia situated at the delta of the Amu Darya river, classically known as the Oxus, which today is spread across Uzbekistan, Kazakhstan, and Turkmenistan. The name of the territory was first mentioned in the Avesta; it is found inscribed at Persepolis and referenced in classical Persian. The name was once transcribed in English as *Khwarezm*, however, the Greek form entered the English lexicon as *Chorasmian*, and this name is used here.

The Chorasmian script is classified into lapidary and cursive forms. The lapidary form is non-joining and occurs on certain specific items, such as a few silver bowls and a flask found in 2005. The cursive Chorasmian form is derived from the lapidary form, and is found on coinage, wooden items, leather, other silver vessels, and ossuaries, and is the form encoded in the Unicode Standard.

Chorasmian contains 21 letters and 7 numbers. The Unicode character names are based on those of Imperial Aramaic characters.

Directionality. The Chorasmian script is a cursively joining *abjad*, most commonly written from right to left, with lines that advance from top to bottom. Some inscriptions are written vertically and read top to bottom with lines that advance from left to right.

Joining Behavior. Letters are classified as dual-joining, right-joining, and non-joining. Dual-joining and right-joining letters have contextual shapes that are determined by adjacent letters. In some cases, a ZWNJ is used to prevent the left-side connection of a dual-joining letter from joining.

Punctuation and Line Breaking. Spaces are used to separate words. There are no special punctuation marks. There are no formal rules to break words at the end of line.

Numbers. The primary numbers one to four are encoded atomically. The numbers five to nine are expressed using combinations of one to four. This model aligns with Imperial Aramaic and related scripts.

10.9 Elymaic

10.9.1 Elymaic: U+10FE0–U+10FFF

The Elymaic script, also called "Elymaean," was used to write Achaemenid Aramaic in the ancient state of Elymais, which flourished from the second century BCE to the early third century CE and was located in the southwestern portion of modern-day Iran. Elymaic derives from the Aramaic script and is closely related to Parthian and Mandaic. The script is found on inscriptions and coins.

Directionality. The Elymaic script is written from right to left. Conformant implementations of the Elymaic script must use the Unicode Bidirectional Algorithm. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Structure. Elymaic is encoded as a non-joining *abjad*. Although some sources show adjacent letters connecting or overlapping, the overall script does not contain intrinsic cursive behavior. However, Elymaic includes one ligature: U+10FF6 ELYMAIC LIGATURE ZAYIN-YODH.

Character Names and Glyphs. The Elymaic character names are based on those for Imperial Aramaic because the native names for the characters are unknown. The representative glyphs in the code charts are based on the stone inscriptions at Tang-e Sarvak in southwest Iran.

Punctuation. There is no script-specific punctuation for Elymaic. Although word boundaries are not generally indicated, some inscriptions have spaces between words. Modern editors tend to use U+0020 SPACE for word separation.

Numerals. There are no known script-specific numerals.

10.10 Nabataean

10.10.1 Nabataean: U+10880–U+108AF

The Nabataean script developed from the Aramaic script and was used to write the language of the Nabataean kingdom. The script was in wide use from the second century BCE to the fourth century CE, well after the Roman province of Arabia Petraea was formed.

Nabataean is generally considered to be the precursor of the Arabic script. The Namara inscription, dating from the fourth century CE and believed to be one of the oldest Arabic texts, was written in the Nabataean script.

The glyphs of the Nabataean script are more ornate than those of other scripts derived from Aramaic, and flourishes can be found in some inscriptions. As the script evolved, a range of ligatures was introduced. Because their usage is irregular, no joining behavior is specified for Nabataean.

Structure. The Nabataean script consists of 22 consonants. Nine consonants have final forms and are treated similarly to the final letters of the Hebrew script. The final forms are encoded separately because their occurrence in text is not predictable. For more information about the use of distinctly encoded final consonants in Semitic scripts, see *Section 9.1, Hebrew*.

Directionality. Both words and numbers in the Nabataean script are written from right to left in horizontal lines. Conformant implementations of the script must use the Unicode Bidirectional Algorithm. For more information on bidirectional layout, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Numerals. Nabataean has script-specific numeral characters, with strong right-to-left directionality. Nabataean numbers are built up using sequences of characters for 1, 2, 3, 4, 5, 10, 20, and 100 in a manner similar to the way numbers are built up for Imperial Aramaic, which is shown in *Table 10-3*. A cruciform variant of the numeral 4 is encoded separately at U+108AB.

Punctuation. There is no script-specific punctuation in Nabataean. The inscriptions usually have no space between words, but modern editors tend to use U+0020 SPACE for word separation.

10.11 Palmyrene

10.11.1 Palmyrene: U+10860–U+1087F

The Palmyrene script was derived by modification of the customary forms of Aramaic developed during the Achaemenid empire. The script was used for writing the Palmyrene dialect of West Aramaic, and is known from inscriptions and documents found mainly in the city of Palmyra and other cities in the region of Syria, dating from 44 BCE to about 280 CE.

Palmyrene has both a monumental and a cursive form. Earlier inscriptions show more rounded forms, while later inscriptions tend to regularize the letterforms. Most pre-Unicode fonts for Palmyrene have followed the monumental style. Ligatures exist in both forms of the script, but are not used consistently.

At a certain point, some Palmyrene letterforms became confused and a distinguishing diacritical dot was introduced, although not regularly or systematically, as seen in the glyphic variation of consonants *daleth* and *resh* across the various styles of the script. Sometimes the two glyphs appear with different skeletons, which is sufficient to distinguish them; sometimes they have the same skeleton and are differentiated by a dot; and sometimes they appear with the same skeleton and no dot, in which case they are indistinguishable. In the Unicode code charts, a dot distinguishes the *daleth* and *resh* glyphs.

Structure. The Palmyrene script consists of 22 consonants. The consonant *nun* has a final form variant, encoded as a separate character, U+1086D PALMYRENE LETTER FINAL NUN, and used similarly to the counterpart Hebrew consonant. For information about the use of distinctly encoded final consonants in Semitic scripts, see *Section 9.1, Hebrew*.

Directionality. Both words and numbers in the Palmyrene script are written from right to left in horizontal lines. Conformant implementations of the script must use the Unicode Bidirectional Algorithm. For more information on bidirectional layout, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Numerals. Palmyrene has script-specific numeral characters, with strong right-to-left directionality. Palmyrene numbers are built up using sequences of characters for 1, 2, 3, 4, 5, 10, 20, and 100 in a manner similar to the way numbers are built up for Imperial Aramaic, which is shown in *Table 10-3*. The glyphs for the numerals 10 and 100, which had been distinct in Aramaic, coalesced into the same glyph in Palmyrene. The two numerals are generally distinguished by their position in sequences representing numbers rather than their shape. A single character is encoded at U+1087E PALMYRENE NUMBER TEN and should be used for both numerals.

Symbols. Two symbols are encoded at U+10877 PALMYRENE LEFT-POINTING FLEURON and U+10878 PALMYRENE RIGHT-POINTING FLEURON. They usually appear next to numbers.

Punctuation. There is no script-specific punctuation in Palmyrene. The inscriptions usually have no space between words, but modern editors tend to use U+0020 SPACE for word separation.

10.12 Hatran

10.12.1 Hatran: U+108E0–U+108FF

The Hatran *abjad* belongs to the North Mesopotamian branch of the Aramaic scripts, and was used for writing a dialect of the Aramaic language. Hatran writing was discovered in the ancient city of Hatra in present-day Iraq. The inscriptions found there date from 98–97 BCE until circa 241 CE, when the city of Hatra was destroyed. Many of the known texts in Hatran are graffiti, but there are some longer texts.

Structure. The Hatran script consists of 22 consonants, encoded as 21 characters. The consonants *daleth* and *resh* are indistinguishable by shape and are encoded as a single character, U+108E3 HATRAN LETTER DALETH-RESH. Ligatures can occur—for example, the letter *beth* often joins or touches the letter following it—but are not used consistently.

Directionality. Both words and numbers in the Hatran script are written from right to left in horizontal lines. Conformant implementations of the script must use the Unicode Bidirectional Algorithm. For more information on bidirectional layout, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Numerals. Hatran has script-specific characters for numerals, with strong right-to-left directionality. Hatran numbers are built up using sequences of characters for 1, 5, 10, 20, and 100 in a manner similar to the way numbers are built up for Imperial Aramaic, which is shown in *Table 10-3*. The numbers 2, 3, and 4 are formed from sequences of repeated characters for the numeral 1, and are not separately encoded.

Punctuation. There is no script-specific punctuation encoded for Hatran. The inscriptions sometimes have spaces between words; modern editors tend to insert U+0020 SPACE for word separation even if there were no spaces in the original text.

Chapter 11

Cuneiform and Hieroglyphs

The following scripts are described in this chapter:

Three ancient cuneiform scripts are described in this chapter: Ugaritic, Old Persian, and Sumero-Akkadian. The largest and oldest of these is Sumero-Akkadian. The other two scripts are not derived directly from the Sumero-Akkadian tradition but had common writing technology, consisting of wedges indented into clay tablets with reed styluses. Ugaritic texts are about as old as the earliest extant Biblical texts. Old Persian texts are newer, dating from the fifth century BCE.

Egyptian Hieroglyphs were used for more than 3,000 years from the end of the fourth millennium BCE.

Meroitic hieroglyphs and Meroitic cursive were used from around the second century BCE to the fourth century CE to write the Meroitic language of the Nile valley kingdom known as Kush or Meroë. Meroitic cursive was for general use, and its appearance was based on Egyptian demotic. Meroitic hieroglyphs were used for inscriptions, and their appearance was based on Egyptian hieroglyphs.

Anatolian Hieroglyphs date to the second and first millennia BCE, and were used to write the Luwian language, an Indo-European language, in the area of present-day Turkey and environs.

11.1 Sumero-Akkadian

Sumero-Akkadian Cuneiform is a logographic writing system with a strong syllabic component. It was written from left to right on clay tablets.

Early History of Cuneiform. The earliest stage of Mesopotamian Cuneiform as a complete system of writing is first attested in Uruk during the so-called Uruk IV period (circa 3500–3200 BCE) with an initial repertoire of about 700 characters or "signs" as Cuneiform scholars customarily call them.

Late fourth millennium ideographic tablets were also found at Susa and several other sites in western Iran, in Assyria at Nineveh (northern Iraq), at Tell Brak (north-western Syria), and at Habuba Kabira in Syria. The writing system developed in Sumer (southeastern Iraq) was repeatedly exported to peripheral regions in the third, second, and first millennia BCE. Local variations in usage are attested, but the core of the system is the Sumero-Akkadian writing system.

Writing emerged in Sumer simultaneously with a sudden growth in urbanization and an attendant increase in the scope and scale of administrative needs. A large proportion of the elements of the early writing system repertoire was devised to represent quantities and commodities for bureaucratic purposes.

At this earliest stage, signs were mainly pictographic, in that a relatively faithful facsimile of the thing signified was traced, although some items were strictly ideographic and represented by completely arbitrary abstractions, such as the symbol for sheep ⊕. Some scholars believe that the abstract symbols were derived from an earlier "token" system of accounting, but there is no general agreement on this point. Where the pictographs are concerned, interpretation was relatively straightforward. The head of a bull was used to denote "cattle"; an ear of barley was used to denote "barley." In some cases, pictographs were also interpreted logographically, so that meaning was derived from the symbol by close conceptual association. For example, the representation of a bowl might mean "bowl," but it could indicate concepts associated with bowls, such as "food." Renditions of a leg might variously suggest "leg," "stand," or "walk."

By the next chronological period of south Mesopotamian history (the Uruk III period, 3200–2900 BCE), logographic usage seems to have become much more widespread. In addition, individual signs were combined into more complex designs to express other concepts. For example, a head with a bowl next to it was used to denote "eat" or "drink." This is the point during script development at which one can truly speak of the first Sumerian texts. In due course, the early graphs underwent change, conditioned by factors such as the most widely available writing medium and writing tools, and the need to record information more quickly and efficiently from the standpoint of the bureaucracy that spawned the system.

Clay was the obvious writing medium in Sumer because it was widely available and easily molded into cushion- or pillow-shaped tablets. Writing utensils were easily made for it by sharpening pieces of reed. Because it was awkward and slow to inscribe curvilinear lines in a piece of clay with a sharpened reed (called a *stylus*), scribes tended to approximate the pictographs by means of short, wedge-shaped impressions made with the edge of the stylus. These short, mainly straight shapes gave rise to the modern word "cuneiform" from the Latin *cuneus*, meaning "wedge." Cuneiform proper was common from about 2700 BCE, although experts use the term "cuneiform" to include the earlier forms as well.

Geographic Range. The Sumerians did not live in complete isolation, and there is very early evidence of another significant linguistic group in the area immediately north of Sumer known as Agade or Akkad. Those peoples spoke a Semitic language whose dialects are subsumed by scholars under the heading "Akkadian." In the long run, the Akkadian speakers became the primary users and promulgators of Cuneiform script. Because of their trade involvement with their neighbors, Cuneiform spread through Babylonia (the umbrella term for Sumer and Akkad) to Elam, Assyria, eastern Syria, southern Anatolia, and even Egypt. Ultimately, many languages came to be written in Cuneiform script, the most notable being Sumerian, Akkadian (including Babylonian and Assyrian), Eblaite, Elamite, Hittite, and Hurrian.

Periods of script usage are defined according to geography and primary linguistic representation, as shown in .

Table 11-1. Cuneiform Script Usage

Archaic Period (to 2901 BCE)			
Early Dynastic (2900–2335 BCE)			
Old Akkadian (2334–2154 BCE)			
Ur III (Neo-Sumerian) (2112–2095 BCE)			Elamite (2100–360 BCE)
Old Assyrian (1900–1750 BCE)		Old Babylonian (2004–1595 BCE)	
Hittite (1570–1220 BCE)	Middle Assyrian (1500–1000 BCE)	Middle Babylonian (1595–627 BCE)	
Neo-Assyrian (1000–609 BCE)			
		Neo-Babylonian (626–539 BCE)	

11.1.1 Cuneiform: U+12000–U+123FF

Coverage. In the Unicode Standard, the Sumero-Akkadian Cuneiform script represents the script used from the Early Dynastic period onwards. In general, signs used in the Ur III period or later have been encoded in the Cuneiform block, whereas signs used solely in earlier periods have been encoded in the Early Dynastic Cuneiform block.

Simple Signs. Most Cuneiform signs are simple units; each sign of this type is represented by a single character in the standard.

Complex and Compound Signs. Some Cuneiform signs are categorized as either complex or compound signs. Complex signs are made up of a primary sign with one of more secondary signs written within it or conjoined to it, such that the whole is generally treated by scholars as a unit; this includes linear sequences of two or more signs or wedge-clusters where one or more of those clusters have not been clearly identified as characters in their own right. Complex signs, which present a relative visual unity, are assigned single individual code points irrespective of their components.

Compound signs are linear sequences of two or more signs or wedge-clusters generally treated by scholars as a single unit, when each and every such wedge-cluster exists as a clearly identified character in its own right. Compound signs are encoded as sequences of their component characters. Signs that shift from compound to complex, or vice versa, generally have been treated according to their Ur III manifestation.

Mergers and Splits. Over the long history of Cuneiform, a number of signs have simplified and merged; in other cases, a single sign has diverged and developed into more than one distinct sign. The choice of signs for encoding as characters was made at the point of maximum differentiation in the case of either mergers or splits to enable the most comprehensive set for the representation of text in any period. In particular, some signs in the main Cuneiform block, while used in the Ur III and later periods, are distinct only in the Early Dynastic period.

Fonts. Fonts for the representation of Cuneiform text need to be designed distinctly for optimal use for different historic periods. The glyphs in the code charts are primarily in the style of the Ur III period, but some are in earlier styles as far back as Early Dynastic, or later styles as late as the first millennium, to illustrate signs specific to these periods or to disambiguate mergers and splits.

Fonts for any period will contain duplicate glyphs depending on the status of merged or split signs at that point of the development of the writing system. These considerations are discussed in greater detail and illustrated in Unicode Technical Report #56, "Unicode Cuneiform Sign Lists."

Glyph Variants Acquiring Independent Semantic Status. Glyph variants such as U+122EC ⿻ cuneiform sign ta asterisk, a Middle Assyrian form of the sign U+122EB ⿻ cuneiform sign ta, which in Neo-Assyrian usage has its own logographic interpretation, have been assigned separate code positions. They are to be used only when the new interpretation applies.

Formatting. Cuneiform was often written between incised lines or in blocks surrounded by drawn boxes known as *case rules*. These boxes and lines are considered formatting and are not part of the script. Case ruling and the like are not to be treated as punctuation.

Other Standards. While there is no standard legacy encoding of Cuneiform, there is a set of well-established conventions for unambiguous transliteration of cuneiform text, as well as standards for the digital representation of these transliterations. The cuneiform encoding, and in particular its handling of mergers and splits, is designed to be compatible with the production of cuneiform text from transliterated corpora. See Unicode Technical Report #56, "Unicode Cuneiform Sign Lists."

Ancillary Data. In practice, implementations of the Sumero-Akkadian Cuneiform script require an association of sequences of code points with entries in the classical sign lists that establish abstract character identity, and with the sign values which provide the usual names of these signs. For more information on such ancillary data, see Unicode Technical Report #56, "Unicode Cuneiform Sign Lists."

11.1.2 Cuneiform Numbers and Punctuation: U+12400–U+1247F

Cuneiform Punctuation. A small number of signs are occasionally used in Cuneiform to indicate word division, repetition, or phrase separation.

Cuneiform Numerals. In general, numerals have been encoded separately from signs that are visually identical but semantically different (for example, U+1244F ⿻ cuneiform numeric sign one ban2, U+12450 ⿻ cuneiform numeric sign two ban2, and so on, versus U+12226 ⿻ cuneiform sign mash, U+1227A ⿻ cuneiform sign pa, and so on).

11.1.3 Early Dynastic Cuneiform: U+12480–U+1254F

This block contains characters covering extensions for Cuneiform for the Early Dynastic period, 2900-2335 BCE. The writing of this period is attested primarily from two sites, Fāra and Tell Abū-Ṣalābīkh, both located in the southern part of Iraq. The attestations include administrative, legal, lexical, and literary texts.

The repertoire in this block is compiled primarily from the modern Assyriological sign list of the Early Dynastic period, *Liste der archaischen Keilschriftzeichen aus Fara* (abbreviated LAK), with a few additions derived from other sources. Only Early

Dynastic signs not already included in the main Cuneiform block have been added here.

11.2 Ugaritic

11.2.1 Ugaritic: U+10380–U+1039F

The city state of Ugarit was an important seaport on the Phoenician coast (directly east of Cyprus, north of the modern town of Minet el-Beida) from about 1400 BCE until it was completely destroyed in the twelfth century BCE. The site of Ugarit, now called Ras Shamra (south of Latakia on the Syrian coast), was apparently continuously occupied from Neolithic times (circa 5000 BCE). It was first uncovered by a local inhabitant while plowing a field in 1928 and subsequently excavated by Claude Schaeffer and Georges Chenet beginning in 1929, in which year the first of many tablets written in the Ugaritic script were discovered. They later proved to contain extensive portions of an important Canaanite mythological and religious literature that had long been sought and that revolutionized Biblical studies. The script was first deciphered in a remarkably short time jointly by Hans Bauer, Edouard Dhorme, and Charles Virolleaud.

The Ugaritic language is Semitic, variously regarded by scholars as being a distinct language related to Akkadian and Canaanite, or a Canaanite dialect. Ugaritic is generally written from left to right horizontally, sometimes using U+1039F ᎞ UGARITIC WORD DIVIDER. In the city of Ugarit, this script was also used to write the Hurrian language. The letters U+1039B ᎒ UGARITIC LETTER I, U+1039C ᎒ UGARITIC LETTER U, and U+1039D ᎒ UGARITIC LETTER SSU are used for Hurrian.

Variant Glyphs. There is substantial variation in glyph representation for Ugaritic. Glyphs for U+10398 ᎒ UGARITIC LETTER THANNA, U+10399 ᎒ UGARITIC LETTER GHAIN, and U+1038F ᎒ UGARITIC LETTER DHAL differ somewhat between modern reference sources, as do some transliterations. U+10398 ᎒ UGARITIC LETTER THANNA is most often displayed with a glyph that looks like an occurrence of U+10393 ᎒ UGARITIC LETTER AIN overlaid with U+10382 ᎒ UGARITIC LETTER GAMLA.

Ordering. The ancient Ugaritic alphabetical order, which differs somewhat from the modern Hebrew order for similar characters, has been used to encode Ugaritic in the Unicode Standard.

Character Names. Some of the Ugaritic character names have been reconstructed; others appear in an early fragmentary document.

11.3 Old Persian

11.3.1 Old Persian: U+103A0–U+103DF

The Old Persian script is found in a number of inscriptions in the Old Persian language dating from the Achaemenid empire. Scholars today agree that the character inventory of Old Persian was invented for use in monumental inscriptions of the Achaemenid king, Darius I, by about 525 BCE. Old Persian is an alphabetic writing system with some syllabic aspects. While the shapes of some Old Persian letters look similar to signs in Sumero-Akkadian Cuneiform, it is clear that only one of them, U+103BE ⟃ OLD PERSIAN SIGN LA, was actually borrowed. It was derived from the New Assyrian historic variant ⟃ of Sumero-Akkadian U+121B7 ⟃ CUNEIFORM SIGN LA, because *la* is a foreign sound not used in the Old Persian language.

Directionality. Old Persian is written from left to right.

Repertoire. The repertoire contains 36 signs. These represent consonants, vowels, or consonant plus vowel syllables. There are also five numbers, one word divider, and eight ideograms. It is considered unlikely that any additional characters will be discovered.

Numerals. The attested numbers are built up by stringing the base numbers (1, 2, 10, 20, and 100) in sequences.

Variants. The signs U+103C8 OLD PERSIAN SIGN AURAMAZDAA and U+103C9 OLD PERSIAN SIGN AURAMAZDAA-2, and the signs U+103CC OLD PERSIAN SIGN DAHYAAUSH and U+103CD OLD PERSIAN SIGN DAHYAAUSH-2, have been encoded separately because their conventional attestation in the corpus of Old Persian texts is quite limited and scholars consider it advantageous to distinguish the forms in plain text representation.

11.4 Egyptian Hieroglyphs

Hieroglyphic writing appeared in Egypt at the end of the fourth millennium BCE. The writing system is pictographic: the glyphs represent tangible objects, most of which modern scholars have been able to identify. A great many of the pictographs are easily recognizable even by nonspecialists. Egyptian hieroglyphs represent people and animals, parts of the bodies of people and animals, clothing, tools, vessels, and so on.

Hieroglyphs were used to write Egyptian for more than 3,000 years, retaining characteristic features such as use of color and detail in the more elaborated expositions. Throughout the Old Kingdom, the Middle Kingdom, and the New Kingdom, between 700 and 1,000 hieroglyphs were in regular use, and there were a large number of rarer hieroglyphs. During the Greco-Roman period, the number of variants, as distinguished by some modern scholars, grew to about 10,000.

Hieroglyphs were carved in stone, painted on frescos, and could also be written with a reed stylus, though this cursive writing eventually became standardized in what is called hieratic writing. The hieratic forms are not separately encoded; they are simply considered cursive forms of the hieroglyphs encoded in this block.

The Demotic script and then later the Coptic script replaced the earlier hieroglyphic and hieratic forms for much practical writing of Egyptian, but hieroglyphs and hieratic continued in use until the fourth century CE. An inscription dated August 24, 394 CE has been found on the Gateway of Hadrian in the temple complex at Philae; this is thought to be among the latest examples of Ancient Egyptian writing in hieroglyphs.

Structure. Egyptian hieroglyphic writing made use of 24 hieroglyphs for individual consonants. Other hieroglyphs are used to represent a sequence of two or three consonants. In addition to these phonetic characters, Egyptian hieroglyphic writing made use of logograms, which could be read as a word or as a classifier, which enables the reader to distinguish between words which were otherwise written the same. Hieroglyphs were arranged next to one another in an aesthetically-pleasing manner, whether horizontally, vertically, or in other arrangements within a notional rectangle. That notional rectangle has traditionally been referred to as a *quadrat*.

Directionality. Characters may be written from left to right or from right to left, either horizontally or vertically. Directionality of a text is usually easy to determine because one reads a line facing into the glyphs depicting the faces of people or animals.

In modern Egyptological publications, arrows are used to indicate whether the hieroglyphic text is laid out horizontally in rows or vertically in columns and the direction

the glyphs are facing. For layout in rows, two arrows are employed: U+2190 ← LEFT-WARDS ARROW and U+2192 → RIGHTWARDS ARROW, with the arrow indicating the direction of the faces. For vertical text, U+2193 ↓ DOWNWARDS ARROW is employed, but that arrow does not specify the direction the hieroglyphs are facing.

For hieroglyphic text written in columns, U+1F8C0 ⃪ LEFTWARDS ARROW FROM DOWNWARDS ARROW is used when the faces are turned towards the left, and U+1F8C1 ⃫ RIGHTWARDS ARROW FROM DOWNWARDS ARROW when the faces are turned towards the right.

Egyptian hieroglyphs are given strong left-to-right directionality in the Unicode Standard, because most contemporary use of Egyptian hieroglyphs uses left-to-right directionality as the presentation mode. When left-to-right directionality is overridden to display Egyptian hieroglyphic text right to left, the glyphs should be mirrored from those shown in the code charts.

Rendering. The encoded characters for Egyptian hieroglyphs in the Unicode Standard simply represent basic text elements, or *signs*, of the writing system. To represent the arrangement of signs horizontally, vertically, or in other positions, a set of format controls should be employed (see "Egyptian Hieroglyph Format Controls").

Hieratic Fonts. In the years since Champollion published his decipherment of Egyptian in 1824, Egyptologists have shown little interest in typesetting hieratic text. Consequently, there is no tradition of hieratic fonts in either lead or digital formats. Because hieratic is a cursive form of the underlying hieroglyphic characters, hieratic text is normally rendered using the more easily legible hieroglyphs, although the hieroglyphic transcription of hieratic text has specific behaviors. (For example, see the discussion of enclosure controls below.) In principle a hieratic font could be devised for specialist applications.

11.4.1 Egyptian Hieroglyphs: U+13000–U+1342F

Repertoire. The set of hieroglyphic characters encoded in the Egyptian Hieroglyphs block is loosely referred to as "the Gardiner set." However, the Gardiner set was not actually exhaustively described and enumerated by Gardiner, himself. The chief source of the repertoire is Gardiner's Middle Egyptian sign list as given in his *Egyptian Grammar* (Gardiner 1957). That list is supplemented by additional characters found in his font catalogues (Gardiner 1928, Gardiner 1929, Gardiner 1931, and Gardiner 1953), and by a collection of signs found in the Griffith Institute's *Topographical Bibliography*, which also used the Gardiner fonts.

A few other characters have been added to this set, such as entities to which Gardiner gave specific catalog numbers. They are retained in the encoding for completeness in representation of Gardiner's own materials. A number of positional variants without catalog numbers were listed in Gardiner 1957 and Gardiner 1928.

Character Names. Egyptian hieroglyphic characters have traditionally been designated in several ways:

- By complex description of the pictographs: GOD WITH HEAD OF IBIS, and so forth.

- By standardized sign number: C3, E34, G16, G17, G24.

- For a minority of characters, by transliterated sound.

The characters in this block use the standard Egyptological catalog numbers for the signs. Thus, the name for U+130F9 ⌖ EGYPTIAN HIEROGLYPH E034 refers uniquely and unambiguously to the Gardiner list sign E34, described as a "DESERT HARE" and used for the sound "wn". The catalog values are padded to three places with zeros.

Names for hieroglyphic characters identified explicitly in Gardiner 1953 or other sources as variants for other hieroglyphic characters are given names by appending "A", "B", ... to the sign number. In the sources these are often identified using asterisks. Thus Gardiner's G7, G7*, and G7** correspond to U+13146 ⌖ EGYPTIAN HIERO-GLYPH G007, U+13147 ⌖ EGYPTIAN HIEROGLYPH G007A, and U+13148 ⌖ EGYPTIAN HIEROGLYPH G007B, respectively.

Sign Classification. In Gardiner's identification scheme, Egyptian hieroglyphs are classified according to letters of the alphabet, so A000 refers to "Man and his occupations," B000 to "Woman and her occupations," C000 to "Anthropomorphic deities," and so forth. The order of signs in the code charts reflects this classification. The Gardiner categories are shown in headers in the names list accompanying the code charts.

Some individual characters may have been identified as belonging to other classes since their original category was assigned, but the ordering in this block of the Unicode Standard simply follows the original category and catalog values.

Enclosures. The two principal names of the king, the nomen and prenomen, were normally written inside a cartouche: a pictographic representation of the name with hieroglyphs that are surrounded by an oval enclosure with a vertical line at one end.

There are a several pairs of characters for the different types of enclosures used in Egyptian hieroglyphic texts. A set of four enclosure controls U+1343C..U+1343F were added in Unicode 15.0 to better represent the different enclosure combinations found in actual text. For examples and details, see the discussion of enclosure controls below.

Numerals. Egyptian numbers are encoded following the same principles used for the encoding of Aegean and Cuneiform numbers. Gardiner does not supply a full set of numerals with catalog numbers in his *Egyptian Grammar*, but does describe the

system of numerals in detail, so that it is possible to deduce the required set of numeric characters.

Two conventions of representing Egyptian numerals are supported in the Unicode Standard. The first relates to the way in which hieratic numerals are represented. Individual signs for each of the 1s, the 10s, the 100s, the 1000s, and the 10,000s are encoded, because in hieratic these are written as units, often quite distinct from the hieroglyphic shapes into which they are transliterated. The other convention is based on the practice of the *Manuel de Codage*, and is comprised of five basic text elements used to build up Egyptian numerals. There is some overlap between these two systems.

11.4.2 Egyptian Hieroglyphs Extended-A: U+13460–U+143FF

This block contains additional Egyptian hieroglyphs, primarily from the Greco-Roman period. Character names in this block are derived algorithmically by prefixing the code point with the string "EGYPTIAN HIEROGLYPH-". Hence the name for U+13460 is EGYPTIAN HIEROGLYPH-13460.

The order of characters in this block follows Gardiner's basic classification (A-Z, Aa), but within each Gardiner category, signs are grouped based on the taxonomy of IFAO (*Institut français d'archéologie orientale*), which is similar to, but not identical with, Gardiner's taxonomy.

For further information on all the hieroglyph characters, including the sources, description, and function of each character, see Unicode Standard Annex #57, "Unicode Egyptian Hieroglyph Database (Unikemet)."

11.4.3 Egyptian Hieroglyph Format Controls: U+13430–U+1345F

The structural arrangement of Egyptian hieroglyphs in notional rectangles or quadrats is handled by format control characters in this block. Ten of the format characters control the basic placement of hieroglyphs in quadrats. They are used to join hieroglyphs vertically, horizontally, as an overlay, or to insert signs into a quadrat. Two format controls are used for grouping signs in complex combinations.

Prior to Version 12.0 of Unicode, many Egyptologists used simple markup conventions to indicate formatting, notably the scheme published in the *Manuel de Codage* (MdC). MdC used ASCII characters to indicate the spatial organization of hieroglyphs. Four of the Egyptian Hieroglyph format controls derive from MdC usage:

- U+13430 ⸫ EGYPTIAN HIEROGLYPH VERTICAL JOINER indicates a vertical join, and corresponds to MdC use of a colon.

- U+13431 ⸬ EGYPTIAN HIEROGLYPH HORIZONTAL JOINER indicates a horizontal join, and corresponds to MdC use of an asterisk.

- U+13437 ◌ EGYPTIAN HIEROGLYPH BEGIN SEGMENT and U+13438 ◌ EGYPTIAN HIEROGLYPH END SEGMENT indicate grouping, and correspond to MdC use of opening and closing parentheses, respectively.

A layout of one hieroglyph above another in the quadrat is represented by inserting U+13430 ◌ EGYPTIAN HIEROGLYPH VERTICAL JOINER between two hieroglyphs, where the first logical glyph in the sequence is the upper of the two hieroglyphs as shown in the first example of *Figure 11-1*. Similarly, U+13431 ◌ EGYPTIAN HIERO-GLYPH HORIZONTAL JOINER joins two adjacent hieroglyphs horizontally. The horizontal ordering of the joined glyphs matches the logical ordering of the two hieroglyphs, as shown in the second example in *Figure 11-1*.

Figure 11-1. Vertical and Horizontal Formatting of Hieroglyphs

Image	Symbolic	Character Sequence
	A1 ◌ O1	<13000, 13430, 13250>
	W24 ◌ Z7	<133CC, 13431, 133F2>

The column labeled "Symbolic" in *Figure 11-1* (and subsequent figures) emulates the way such quadrats are represented using the MdC conventions. Thus "A1" is the symbolic abbreviation used in MdC for U+13000 𓀀 EGYPTIAN HIEROGLYPH A001 (a seated man). MdC simply uses a few ASCII characters (":", "*", "+") for the operators that combine signs into sequences expressing the full quadrats. So, the MdC representation of the first example in *Figure 11-1* would be "A1:O1". The symbolic representation in *Figure 11-1* instead uses the dotted box glyph convention to represent the actual Unicode Egyptian Hieroglyph format controls, as for example, U+13430 ◌ EGYPTIAN HIEROGLYPH VERTICAL JOINER.

Four control characters are used in similar fashion to insert a following hieroglyph into the corner of a preceding hieroglyph:

- U+13432 ◌ EGYPTIAN HIEROGLYPH INSERT AT TOP START places a following hieroglyph within the frame of the preceding hieroglyph in the corner at the top edge and starting side.

- U+13433 ◌ EGYPTIAN HIEROGLYPH INSERT AT BOTTOM START causes a following hieroglyph to display in the bottom-starting corner within the frame of the preceding hieroglyph.

- U+13434 ⬚ EGYPTIAN HIEROGLYPH INSERT AT TOP END causes a following hieroglyph to display in the top-ending corner within the frame of the preceding hieroglyph.

- U+13435 ⬚ EGYPTIAN HIEROGLYPH INSERT AT BOTTOM END causes a following hieroglyph to display in the bottom-ending corner within the frame of the preceding hieroglyph.

The first four rows of *Figure 11-2* show examples of this use.

Figure 11-2. Insertion and Overlay Formatting of Hieroglyphs

Image	Symbolic	Character Sequence
	F4 ⬚ X1	<13102, 13432, 133CF>
	I10 ⬚ A1	<13193, 13433, 13000>
	D17 ⬚ X1	<13087, 13434, 133CF>
	G25 ⬚ X1	<1315C, 13435, 133CF>
	D36 ⬚ V28	<1309D, 13436, 1339B>
	N37 ⬚ O34	<13219, 13439, 13283>
	D28 ⬚ J1	<13093, 1343A, 1340D>
	D32 ⬚ V28	<13098, 1343B, 1339B>

U+13439 ⬚ EGYPTIAN HIEROGLYPH INSERT AT MIDDLE is employed to insert a sign in the middle of another. Note that when inserting into the HWT enclosure, only a single group of one or more signs can be inserted. If a sequence of groups is to be enclosed into the HWT, the enclosure controls should be used, as described later in this section under "Enclosure Controls." When signs appear within another hieroglyph that has an opening above, U+1343A ⬚ EGYPTIAN HIEROGLYPH INSERT AT TOP is employed, and for signs that appear within a hieroglyph with an opening below, U+1343B ⬚ EGYPTIAN HIEROGLYPH INSERT AT BOTTOM is used, as shown in the bottom two examples in *Figure 11-2*.

Orthographic checking should handle cases where there may be ambiguity in the encoding choice, such as a choice between insert at middle versus insert at bottom.

When an insertion is to be used with a sign without a clear space to receive the insertion, font developers may use a ligature or alternate glyph to render the expected form, as shown in *Figure 11-3*.

Figure 11-3. Use of U+13439 to Insert at Middle

O13 ▣ R4   <13267, 13439, 132B5>

Hieroglyphs may also overlay other hieroglyphs. This arrangement is controlled by U+13436 ⊞ EGYPTIAN HIEROGLYPH OVERLAY MIDDLE. This control character causes a following hieroglyph to overlay on top of a preceding hieroglyph, as shown in the fifth example in *Figure 11-2*. Glyphs that overlay one another stack at their center points.

Enclosure Controls. A set of four enclosure controls encoded in the range U+1343C..U+1343F represent the different combinations of enclosures that occur in hieroglyphic text. As shown in the upper left example in *Figure 11-4*, the combination of the enclosures and the enclosure controls creates a full-form enclosing cartouche with horizontal lines above and below. The begin and end enclosure format controls must be used in pairs: U+1343C and U+1343D, or U+1343E and U+1343F in the case of walled enclosures.

Figure 11-4. Rendering Enclosures

Egyptian Hieroglyphs	Hieratic Transcription
  <13379, 1343C, 131F3, 13124, 13093, 1343D, 1337A>	 <13379, 131F3, 13124, 13093, 1337A>

Horizontal lines do not appear in cartouches in hieratic text, so the enclosure controls should not be used. An example is shown on the right in *Figure 11-4*. If the enclosure controls are not present, the enclosure characters will appear as stand-alone characters. In the case of damaged text, one or both ends of the cartouche may be missing.

Complex Clusters. The basic joining controls may be used in conjunction with one another to render more complex clusters, as shown in the first example in *Figure 11-5*.

The two characters, U+13437 ⦅ EGYPTIAN HIEROGLYPH BEGIN SEGMENT and U+13438 ⦆ EGYPTIAN HIEROGLYPH END SEGMENT, are used to group signs in

complex clusters comprising different levels of joining controls, as shown in the second example in <u>*Figure 11-5*</u>.

Some rendering systems may support multiple levels of the segment controls for use in the most complex hieroglyphic sign arrangements, as shown in the third example in <u>*Figure 11-5*</u>.

Figure 11-5. Complex Cluster Formatting of Hieroglyphs

Image	Symbolic	Character Sequence
	I9 ⠒ N35 ⠒ F20 ⌐ A1	<13191, 13430, 13216, 13430, 13113, 13433, 13000>
	G9 ⌐ ⦇ N27 ⠒ N27 ⦈	<1314A, 13434, 13437, 1320C, 13430, 1320C, 13438>
	J15 ⠒ Z11 ✳ ⦇ D2 ✳ ⦇ D21 ⠒ X1 ⦈ ⠒ N25 ⦈	<1341D, 13430, 133F6, 13431, 13437, 13077, 13431, 13437, 1308B, 13430, 133CF, 13438, 13430, 13209, 13438>

Some Egyptian hieroglyphs with complex structures are encoded as single characters. The guidance on whether to use the complex characters has evolved over time: complex characters were at first systematically recommended, then later systematically recommended against. This guidance has since become more nuanced. The current best practice is to use a complex character when it conveys a function that is not covered by the meaning of its individual parts, but to use a sequence of atomic signs joined with formatting controls when the function of the compound is covered by the meaning of the atomic signs. Whenever sequences are preferred over a complex character, font designers should include ligatures for these sequences so that they render well.

For example, U+13217 ☰ EGYPTIAN HIEROGLYPH N035A looks like a stack of three copies of U+13216 ☰ EGYPTIAN HIEROGLYPH N035 and could be represented by the sequence <13216, 13430, 13216, 13430, 13216>. However, this compound sign is a logograph for the word for water, *mw*, whereas the parts are phonemograms with the unrelated value *n*. As a result, the atomic character U+13217 is preferred. In contrast, consider U+130C1 ⌐ EGYPTIAN HIEROGLYPH D059, which looks like U+1309D ⌐ EGYPTIAN HIEROGLYPH D036 over U+130C0 ⌐ EGYPTIAN HIEROGLYPH D058, so that it can be represented as the sequence <130C0, 13436, 1309D>. U+130C1 ⌐ is a phonemogram with the value *ꜥb*, and the parts are phonemograms whose value make up *ꜥb* — U+1309D ⌐ has the value *ꜥ* and U+130C0 ⌐ has the value *b*. In this case, the sequence <130C0, 13436, 1309D> is preferred. For information on the function and value of an individual hieroglyph, as well as descriptions of complex hieroglyphs in terms of atomic parts, see Unicode Standard Annex #57, "Unicode Egyptian Hieroglyph Database (Unikemet)."

Mirroring. Scribes frequently mirrored individual signs for symmetry or in cartouches. The format control character U+13440 ⬚ EGYPTIAN HIEROGLYPH MIRROR HORIZONTALLY can be used to mirror a sign. Mirroring is based on the line direction, and the use of this formatting character is independent of any mirroring produced by changing the base direction of the text.

U+13440 should not be used if mirroring would change the meaning of the sign; the separately encoded character should be used instead. For example, the logogram U+130BB ⬚ EGYPTIAN HIEROGLYPH D054 is used for "come," but U+130BD ⬚ EGYPTIAN HIEROGLYPH D055 is a determinative for "going backwards," and that character should be used rather than mirroring.

Signs that are horizontally symmetrical do not require mirroring, and fonts might render U+13440 ⬚ EGYPTIAN HIEROGLYPH MIRROR HORIZONTALLY visibly in such contexts.

Rotation. A rotated sign that has a distinct meaning from the unrotated sign should be encoded as a separate character. The separately encoded rotated character should be employed in such contexts, rather than using a variation sequence for rotation.

Rotations of signs are defined in a set of standardized variation sequences in StandardizedVariants.txt in the Unicode Character Database. In combination with Egyptian Hieroglyphs, U+FE00 VARIATION SELECTOR-1 (VS1) is used to request a 90 degree rotation, U+FE01 (VS2) marks a 180 degree rotation and U+FE02 (VS3) is used for 270 degree rotation, as shown in the first row of *Figure 11-6*. For text that runs from left to right, the direction of rotation is clockwise, while it is counterclockwise for text that runs from right to left, as shown in the second row of *Figure 11-6*. If a sign is both rotated and mirrored, rotation is done before mirroring.

Figure 11-6. Rotation of Hieroglyphs

Direction	0°	90°	180°	270°
→				
	<13000>	<13000, FE00>	<13000, FE01>	<13000, FE02>
←				
	<13000>	<13000, FE00>	<13000, FE01>	<13000 , FE02>

For glyphs that are symmetrical, a 90° rotation and a 270° rotation may have the same visual result. For example, U+13399 ⌣ EGYPTIAN HIEROGLYPH V026 is horizontally positioned, but can be rotated to be vertical either with a 90° rotation or a 270° rotation. In such cases, only one sequence is defined in StandardizedVariants.txt.

11.4.4 Editorial Marks

Blanks. To represent an empty surface that never contained any text, U+13441 ▫ EGYPTIAN HIEROGLYPH FULL BLANK and U+13442 ▫ EGYPTIAN HIEROGLYPH HALF BLANK characters are employed. A blank character is used, for example, when a scribe intended to fill in a name or date later, but never filled in the space with text. The blanks are rendered as whitespace, as shown in *Figure 11-7*.

Figure 11-7. Use of Blanks

Lost Signs. To indicate text that had existed earlier, but was later destroyed, U+13443 ■ EGYPTIAN HIEROGLYPH LOST SIGN, U+13444 ▪ EGYPTIAN HIEROGLYPH HALF LOST SIGN, U+13445 ▐ EGYPTIAN HIEROGLYPH TALL LOST SIGN and U+13446 ▬ EGYPTIAN HIEROGLYPH WIDE LOST SIGN are used. Some of these lost signs are shown in *Figure 11-8* next to other extant signs. The "lost signs" may appear in groups with other signs and are generally rendered as shaded squares or rectangles with whitespace between the signs. If continuous shading is required without whitespace between the signs, then U+FE00 VARIATION SELECTOR-1 immediately follows the blank lost character, so that no whitespace appears.

Figure 11-8. Use of Lost Signs

Damage Modifiers. Damaged portions of text are handled by a series of 15 modifiers (U+13447..U+13455). The surface is divided into four quarters, with a single modifier indicating which quarters are damaged. When the entire space is damaged, U+13455 ▧ EGYPTIAN HIEROGLYPH MODIFIER DAMAGED should be employed, as shown in the final example in *Figure 11-9*.

Figure 11-9. Damage Modifiers for Hieroglyphs

G1 ▪️   <1313F, 13447>

I10 ▮   <13193, 13452>

G25 ◾   <1315C, 13454>

L1 ■   <131A3, 13455>

Text Critical Marks. Modern scholarship uses a variety of brackets to indicate notable features of a text, especially destruction and emendation. *Table 11-2* illustrates the commonly used signs that may be used with Egyptian hieroglyphs. These signs are placed logically before and after the sign or group of signs they modify. Implementors should pay particular attention to make sure these signs are supported in fonts and can participate in quadrat structures.

Table 11-2. Brackets used with Egyptian Hieroglyphs

Signs	Code points	Function
[ ]	U+005B LEFT SQUARE BRACKET, U+005D RIGHT SQUARE BRACKET	Complete destruction of a sign or signs
⌐¬	U+2E22 TOP LEFT HALF BRACKET, U+2E23 TOP RIGHT HALF BRACKET	Partial destruction of a sign or signs
⟨⟩	U+27E8 MATHEMATICAL LEFT ANGLE BRACKET, U+27E9 MATHEMATICAL RIGHT ANGLE BRACKET	Modern emendation, addition
{ }	U+007B LEFT CURLY BRACKET, U+007D RIGHT CURLY BRACKET	Modern emendation, deletion
⟦ ⟧	U+27E6 MATHEMATICAL LEFT WHITE SQUARE BRACKET, U+27E7 MATHEMATICAL RIGHT WHITE SQUARE BRACKET	Ancient erasure/deletion

Figure 11-10 illustrates the use of the square brackets to denote signs that are destroyed in the original context but have been reconstructed by a modern editor. The complex quadrat with bracketing in *Figure 11-10* is represented by the sequence <131A3, 005B, 1308B, 13430, 133CF, 13431, 005B 133E5, 13437, 1339F 13430, 133CF, 13438, 005D>.

Figure 11-10. Use of Square Brackets with Hieroglyphs
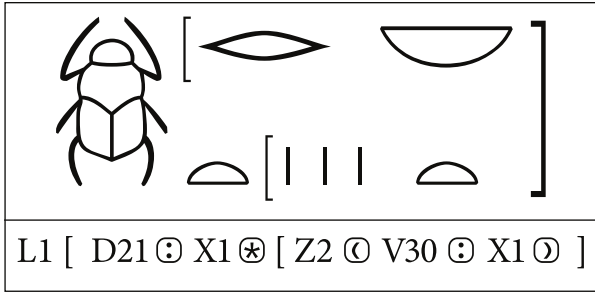
11.5 Meroitic

11.5.1 Meroitic Hieroglyphs: U+10980–U+1099F

Meroitic Cursive: U+109A0–U+109FF

Meroitic hieroglyphs and Meroitic cursive were used from around the second century BCE to the fourth century CE to write the Meroitic language of the Nile valley kingdom known as Kush or Meroë. The kingdom originated south of Egypt around 850 BCE, with its capital at Napata, located in modern-day northern Sudan. At that time official inscriptions used the Egyptian language and script. Around 560 BCE the capital was relocated to Meroë, about 600 kilometers upriver. As the use of Egyptian language and script declined with the greater distance from Egypt, two native scripts developed for writing Meroitic:

- Meroitic cursive was for general use, and its appearance was based on Egyptian demotic.
- Meroitic hieroglyphs were used for inscriptions on royal monuments and temples, and their appearance was based on Egyptian hieroglyphs. (See *Section 11.4, Egyptian Hieroglyphs* for more information.)

After the fourth century CE, the Meroitic language was gradually replaced by Nubian, and by the sixth century the Meroitic scripts had been superseded by the Coptic script, which picked up three additional symbols from Meroitic cursive to represent Nubian.

Although the values of the script characters were deciphered around 1911 by the English Egyptologist F. L. Griffith, the Meroitic language is still not understood except for names and a few other words. It is not known to be related to any other language. It may be related to Nubian.

Structure. Unlike the Egyptian scripts, the Meroitic scripts are almost purely alphabetic. There are 15 basic consonants; if not followed by an explicit vowel letter, they are read with an inherent *a*. There are four vowels: *e*, *i*, *o*, and *a*. The *a* vowel is only used for initial *a*. In addition, for unknown reasons, there are explicit letters for the syllables *ne*, *te*, *se*, and *to*. This may have been due to dialect differences, or to the possible use of *n*, *t*, and *s* as final consonants in some cases.

Meroitic cursive also uses two logograms for *rmt* and *imn*, derived from Egyptian demotic.

Directionality. Horizontal writing is almost exclusively right-to-left, matching the direction in which the hieroglyphs depicting people and animals are looking. This is

unlike Egyptian hieroglyphs, which are read into the faces of the glyphs for people and animals. Meroitic hieroglyphs are also written vertically in columns.

Shaping. In Meroitic cursive, the letter for *i* usually connects to a preceding consonant. There is no other connecting behavior.

Punctuation. The Meroitic scripts were among the earliest to use word division—not always consistently—to separate basic sentence elements, such as noun phrases, verb forms, and so on. For this purpose Meroitic hieroglyphs use three vertical dots, represented by U+205D TRICOLON. When Meroitic hieroglyphs are presented in vertical columns, the orientation of the three dots shifts to become three horizontal dots. This can be represented either with U+2026 HORIZONTAL ELLIPSIS, or in more sophisticated rendering, by glyphic rotation of U+205D TRICOLON. Meroitic cursive uses two vertical dots, represented by U+003A COLON.

Symbols. Two ankh-like symbols are used with Meroitic hieroglyphs.

Meroitic Cursive Numbers. Meroitic numbers are found only in Meroitic Cursive. The system consists of numbers one through nine and bases for ranks: tens, hundreds, thousands, ten thousands, and hundred thousands. The numbers for 100 and higher are systematically formed by attaching the numbers for one through nine as a multiplier to the respective base for each rank. There is also a notation for a fractional system based on twelfths, which simply uses one to eleven dots to represent each fraction.

11.6 Anatolian Hieroglyphs

11.6.1 Anatolian Hieroglyphs: U+14400–U+1467F

Anatolian hieroglyphs appeared on personal seals, monumental inscriptions, and other objects in the second and first millennia BCE in present-day Turkey and surrounding areas. The script, known also as Luwian or Luvian hieroglyphs, was used primarily to write the Luwian language.

Structure. Anatolian hieroglyphs contain both syllabic and logographic elements. Words can be represented by logographs alone, by logographs with a phonetic complement, or solely by syllabic values.

Directionality. Anatolian hieroglyphs can be written from left to right, from right to left, or boustrophedon, and lines are often divided by horizontal rules. Within a line, characters are grouped vertically, typically from top to bottom, although the characters may be placed out of phonetic or logical order for aesthetic reasons.

The characters in the Anatolian Hieroglyphs block have a strong left-to-right directionality (Bidi_Class = L), because publications typically lay out hieroglyphs from left to right. When Anatolian hieroglyphs are displayed right to left, the glyphs should be mirrored from those shown in the code charts.

Repertoire. The repertoire of characters is broadly based on the sign catalog of Laroche (1960), supplemented by additions from later handbooks. Some signs contained in Laroche are considered variants today, but have been encoded separately to represent the complete history of Anatolian scholarship and discussions about the decipherment.

Character names for variant signs are usually distinguished by an "A", "B", or "C" appended to the catalog number of the main sign. For example, U+14600 ⚆ ANATOLIAN HIEROGLYPH A457A is a variant of U+145FF ⚇ ANATOLIAN HIEROGLYPH A457.

A few hieroglyphs developed a simplified, cursive shape, based on the more pictorial shape of the signs found on monuments. The simplified forms are encoded separately, and are differentiated in their names.

1442B	✍	ANATOLIAN HIEROGLYPH A041	*(monumental style)*
		= capere	
		= syllabic tà	
1442C	✍	ANATOLIAN HIEROGLYPH A041A	*(cursive style)*
		= syllabic tà	

The script contains a productive grapheme, U+145B1 ╲ ANATOLIAN HIEROGLYPH A383 RA OR RI, which appears as a part of several other signs, such as U+145B9 ⫴⊢ ANATO-

LIAN HIEROGLYPH A389. The characters containing this graphic element as part of their form are not decomposable.

Annotations. Latin names are used traditionally to describe characters used logographically and appear as annotations in the names list. Those characters which have a Luwian phonetic value or are logosyllabic are identified in the annotations. When a plus sign appears between two elements in the annotation, the elements are considered a single graphic unit, whereas a period between the two elements indicates the two elements are considered graphically separate.

1447E	🐂	ANATOLIAN HIEROGLYPH A107
		= bos+mi
14480	🐂	ANATOLIAN HIEROGLYPH A107B
		= bos.mi

Punctuation. In some texts, word division is indicated by U+145B5 ⵏ ANATOLIAN HIEROGLYPH A386 or its variant U+145B6 ⵎ ANATOLIAN HIEROGLYPH A386A. U+145CE ⵢ ANATOLIAN HIEROGLYPH A410 BEGIN LOGOGRAM MARK and U+145CF ⵣ ANATOLIAN HIEROGLYPH A410A END LOGOGRAM MARK sometimes occur in text to mark logograms.

The characters U+145F7 ⵊ ANATOLIAN HIEROGLYPH A450 and U+144EF ⵉ ANATOLIAN HIEROGLYPH A209 are occasionally used to fill blank spaces, often at the end of a word. Spaces are used in modern renditions of hieroglyphic text.

Numbers. Some of the hieroglyphic signs have been interpreted as having numeric values. These include values for 1–5, 8–10, 12, 100, and 1000. However, all of the Anatolian hieroglyphs have the General_Category = Other_Letter and no specific numeric values for them are assigned in the Unicode Character Database.

Rendering. Just as for Egyptian hieroglyphs, only the basic text elements of the script are encoded. A higher-level protocol is required for the display Anatolian hieroglyphs in a nonlinear layout.

Chapter 12

South and Central Asia-I
Official Scripts of India

The scripts of South Asia share so many common features that a side-by-side comparison of a few will often reveal structural similarities even in the modern letterforms. With minor historical exceptions, they are written from left to right. They are all *abugidas* in which most symbols stand for a consonant plus an inherent vowel (usually the sound /a/). Word-initial vowels in many of these scripts have distinct symbols, and word-internal vowels are usually written by juxtaposing a vowel sign in the vicinity of the affected consonant. Absence of the inherent vowel, when that occurs, is frequently marked with a special sign. In the Unicode Standard, this sign is denoted by the Sanskrit word *virāma*. In some languages, another designation is preferred. In Hindi, for example, the word *hal* refers to the character itself, and *halant* refers to the consonant that has its inherent vowel suppressed; in Tamil, the word *puḷḷi* is used. The virama sign nominally serves to suppress the inherent vowel of the consonant to which it is applied; it is a combining character, with its shape varying from script to script.

Most of the scripts of South Asia, from north of the Himalayas to Sri Lanka in the south, from Pakistan in the west to the easternmost islands of Indonesia, are derived from the ancient Brahmi script. The oldest lengthy inscriptions of India, the edicts of Ashoka from the third century BCE, were written in two scripts, Kharoshthi and Brahmi. These are both ultimately of Semitic origin, probably deriving from Aramaic, which was an important administrative language of the Middle East at that time. Kharoshthi, written from right to left, was supplanted by Brahmi and its derivatives. The descendants of Brahmi spread with myriad changes throughout the subcontinent and outlying islands. There are said to be some 200 different scripts deriving from it. By the eleventh century, the modern script known as Devanagari was in ascendancy in India proper as the major script of Sanskrit literature.

The North Indian branch of scripts was, like Brahmi itself, chiefly used to write Indo-European languages such as Pali and Sanskrit, and eventually the Hindi, Bangla, and Gujarati languages, though it was also the source for scripts for non-Indo-European languages such as Tibetan, Mongolian, and Lepcha.

The South Indian scripts are also derived from Brahmi and, therefore, share many structural characteristics. These scripts were first used to write Pali and Sanskrit but

were later adapted for use in writing non-Indo-European languages—namely, the languages of the Dravidian family of southern India and Sri Lanka. Because of their use for Dravidian languages, the South Indian scripts developed many characteristics that distinguish them from the North Indian scripts. South Indian scripts were also exported to southeast Asia and were the source of scripts such as Tai Tham (Lanna) and Myanmar, as well as the insular scripts of the Philippines and Indonesia.

The shapes of letters in the South Indian scripts took on a quite distinct look from the shapes of letters in the North Indian scripts. Some scholars suggest that this occurred because writing materials such as palm leaves encouraged changes in the way letters were written.

The major official scripts of India proper, including Devanagari, are documented in this chapter. They are all encoded according to a common plan, so that comparable characters are in the same order and relative location. This structural arrangement, which facilitates transliteration to some degree, is based on the Indian national standard (ISCII) encoding for these scripts.

The first six columns in each script are isomorphic with the ISCII-1988 encoding, except that the last 11 positions (U+0955..U+095F in Devanagari, for example), which are unassigned or undefined in ISCII-1988, are used in the Unicode encoding. The seventh column in each of these scripts, along with the last 11 positions in the sixth column, represent additional character assignments in the Unicode Standard that are matched across some or all of the scripts. For example, positions U+xx66..U+xx6F and U+xxE6..U+xxEF code the Indic script digits for each script. The eighth column for each script is reserved for script-specific additions that do not correspond from one Indic script to the next.

While the arrangement of the encoding for the scripts of India is based on ISCII, this does not imply that the rendering behavior of South Indian scripts in particular is the same as that of Devanagari or other North Indian scripts. Implementations should ensure that adequate attention is given to the actual behavior of those scripts; they should not assume that they work just as Devanagari does. Each block description in this chapter describes the most important aspects of rendering for a particular script as well as unique behaviors it may have.

Many of the character names in this group of scripts represent the same sounds, and common naming conventions are used for the scripts of India.

12.1 Devanagari

12.1.1 Devanagari: U+0900–U+097F

The Devanagari script is used for writing classical Sanskrit and its modern historical derivative, Hindi. Extensions to the Sanskrit repertoire are used to write other related languages of India (such as Marathi) and of Nepal (Nepali). In addition, the Devanagari script is used to write the following languages: Awadhi, Bagheli, Bhatneri, Bhili, Bihari, Braj Bhasha, Chhattisgarhi, Garhwali, Gondi (Betul, Chhindwara, and Mandla dialects), Harauti, Ho, Jaipuri, Kachchhi, Kanauji, Konkani, Kului, Kumaoni, Kurku, Kurukh, Marwari, Mundari, Newari, Palpa, and Santali.

All other Indic scripts, as well as the Sinhala script of Sri Lanka, the Tibetan script, and the Southeast Asian scripts, are historically connected with the Devanagari script as descendants of the ancient Brahmi script. The entire family of scripts shares a large number of structural features.

The principles of the Indic scripts are covered in some detail in this introduction to the Devanagari script. The remaining introductions to the Indic scripts are abbreviated but highlight any differences from Devanagari where appropriate.

Standards. The Devanagari block of the Unicode Standard is based on ISCII-1988 (Indian Script Code for Information Interchange). The ISCII standard of 1988 differs from and is an update of earlier ISCII standards issued in 1983 and 1986.

The Unicode Standard encodes Devanagari characters in the same relative positions as those coded in positions A0–F4$_{16}$ in the ISCII-1988 standard. The same character code layout is followed for eight other Indic scripts in the Unicode Standard: Bengali/Bangla, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, and Malayalam. This parallel code layout emphasizes the structural similarities of the Brahmi scripts and follows the stated intention of the Indian coding standards to enable one-to-one mappings between analogous coding positions in different scripts in the family. Sinhala, Tibetan, Thai, Lao, Khmer, Myanmar, and other scripts depart to a greater extent from the Devanagari structural pattern, so the Unicode Standard does not attempt to provide any direct mappings for these scripts to the Devanagari order.

In November 1991, at the time *The Unicode Standard, Version 1.0,* was published, the Bureau of Indian Standards published a new version of ISCII in Indian Standard (IS) 13194:1991. This new version partially modified the layout and repertoire of the ISCII-1988 standard. Because of these events, the Unicode Standard does not precisely follow the layout of the current version of ISCII. Nevertheless, the Unicode Standard remains a superset of the ISCII-1991 repertoire. Modern, non-Vedic texts encoded with ISCII-1991 may be automatically converted to Unicode code points and back to their original encoding without loss of information. The Vedic extension

characters defined in IS 13194:1991 *Annex G—Extended Character Set for Vedic* are now fully covered by the Unicode Standard, but the conversions between ISCII and Unicode code points in some cases are more complex than for modern texts.

Encoding Principles. The writing systems that employ Devanagari and other Indic scripts constitute *abugidas*—a cross between syllabic writing systems and alphabetic writing systems. The effective unit of these writing systems is the orthographic syllable, consisting of a consonant and vowel (CV) core and, optionally, one or more preceding consonants, with a canonical structure of (((C)C)C)V. The orthographic syllable need not correspond exactly with a phonological syllable, especially when a consonant cluster is involved, but the writing system is built on phonological principles and tends to correspond quite closely to pronunciation.

The orthographic syllable is built up of alphabetic pieces, the actual letters of the Devanagari script. These pieces consist of three distinct character types: consonant letters, independent vowels, and dependent vowel signs. In a text sequence, these characters are stored in logical (phonetic) order. Consonant letters by themselves constitute a CV unit, where the V is an *inherent* vowel, whose exact phonetic value may vary by writing system. Independent vowels also constitute a CV unit, where the C is considered to be null.

A dependent vowel sign is used to represent a V in CV units where C is not null and V is not the inherent vowel. CV units are not represented by sequences of a consonant followed by virama followed by independent vowel. In some cases, a phonological diphthong (such as Hindi जाओ /jāo/) is actually written as two orthographic CV units, where the second of these units is an independent vowel letter, whose C is considered to be null.

12.1.2 Principles of the Devanagari Script

Rendering Devanagari Characters. Devanagari characters, like characters from many other scripts, can combine or change shape depending on their context. A character's appearance is affected by its ordering with respect to other characters, the font used to render the character, and the application or system environment. These variables can cause the appearance of Devanagari characters to differ from their nominal glyphs (used in the code charts).

Additionally, a few Devanagari characters cause a change in the order of the displayed characters. This reordering is not commonly seen in non-Indic scripts and occurs independently of any bidirectional character reordering that might be required.

Consonant Letters. Each consonant letter represents a single consonantal sound but also has the peculiarity of having an *inherent vowel*, generally the short vowel /a/ in Devanagari and the other Indic scripts. Thus U+0915 DEVANAGARI LETTER KA repre-

sents not just /k/ but also /ka/. In the presence of a dependent vowel, however, the inherent vowel associated with a consonant letter is overridden by the dependent vowel.

Consonant letters may also be rendered as *half-forms*, which are presentation forms used within an orthographic syllable to depict initial consonants in a consonant cluster. These half-forms do not have an inherent vowel. Their rendered forms in Devanagari often resemble the full consonant but are missing the vertical stem, which marks a syllabic core. The stem glyph is graphically and historically related to the sign denoting the inherent /a/ vowel, as discussed later in this section.

Some Devanagari consonant letters have alternative presentation forms whose choice depends on neighboring consonants. This variability is especially notable for U+0930 DEVANAGARI LETTER RA, which has numerous different forms, both as the initial element and as the final element of a consonant cluster. Only the nominal forms, rather than the contextual alternatives, are depicted in the code charts.

The traditional Sanskrit/Devanagari alphabetic encoding order for consonants follows articulatory phonetic principles, starting with velar consonants and moving forward to bilabial consonants, followed by liquids and then fricatives. ISCII and the Unicode Standard both observe this traditional order.

Independent Vowel Letters. The independent vowels in Devanagari are letters that stand on their own. The writing system treats independent vowels as orthographic CV syllables in which the consonant is null. The independent vowel letters are used to write syllables that start with a vowel.

Dependent Vowel Signs (Matras). The dependent vowels serve as the common manner of writing noninherent vowels and are generally referred to as *vowel signs*, or as *matras* in Sanskrit. The dependent vowels do not stand alone; rather, they are visibly depicted in combination with a base letterform. A single consonant or a consonant cluster may have a dependent vowel applied to it to indicate the vowel quality of the syllable, when it is different from the inherent vowel. Explicit appearance of a dependent vowel in a syllable overrides the inherent vowel of a single consonant letter.

The greatest variation among different Indic scripts is found in the way that the dependent vowels are applied to base letterforms. Devanagari has a collection of nonspacing dependent vowel signs that may appear above or below a consonant letter, as well as spacing dependent vowel signs that may occur to the right or to the left of a consonant letter or consonant cluster. Other Indic scripts generally have one or more of these forms, but what is a nonspacing mark in one script may be a spacing mark in another. Also, some of the Indic scripts have single dependent vowels that are indicated by two or more glyph components—and those glyph components may

surround a consonant letter both to the left and to the right or may occur both above and below it.

In modern usage the Devanagari script has only one character denoting a left-side dependent vowel sign: U+093F DEVANAGARI VOWEL SIGN I. In the historic Prishthamatra orthography, Devanagari also made use of one additional left-side dependent vowel sign: U+094E DEVANAGARI VOWEL SIGN PRISHTHAMATRA E. Other Indic scripts either have no such vowel signs (Telugu and Kannada) or include as many as three of these signs (Bengali/Bangla, Tamil, and Malayalam).

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-1* shows vowel letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-1. Devanagari Vowel Letters

For	Use	Do Not Use
ऄ	0904	<0905, 0946>
आ	0906	<0905, 093E>
ई	0908	<0930, 094D, 0907>
ऊ	090A	<0909, 0941>
ऍ	090D	<090F, 0945>
ऎ	090E	<090F, 0946>
ऐ	0910	<090F, 0947>
ऑ	0911	<0905, 0949> or <0906, 0945>
ऒ	0912	<0905, 094A> or <0906, 0946>
ओ	0913	<0905, 094B> or <0906, 0947>
औ	0914	<0905, 094C> or <0906, 0948>
ऒ	0972	<0905, 0945>
अ	0973	<0905, 093A>
आ	0974	<0905, 093B> or <0906, 093A>
औ	0975	<0905, 094F>
अ	0976	<0905, 0956>
अ	0977	<0905, 0957>

Virama (Halant). Devanagari employs a sign known in Sanskrit as the *virama* or vowel omission sign. In Hindi, it is called *hal* or *halant*, and that term is used in

referring to the virama or to a consonant with its vowel suppressed by the virama. The terms are used interchangeably in this section.

The virama sign, U+094D DEVANAGARI SIGN VIRAMA, nominally serves to cancel (or kill) the inherent vowel of the consonant to which it is applied. When a consonant has lost its inherent vowel by the application of virama, it is known as a *dead consonant*; in contrast, a *live consonant* is one that retains its inherent vowel or is written with an explicit dependent vowel sign. In the Unicode Standard, a dead consonant is defined as a sequence consisting of a consonant letter followed by a virama. The default rendering for a dead consonant is to position the virama as a combining mark bound to the consonant letterform.

For example, if C_n denotes the nominal form of consonant C, and C_d denotes the dead consonant form, then a dead consonant is encoded as shown in *Figure 12-1*.

Figure 12-1. Dead Consonants in Devanagari

$$TA_n + VIRAMA_n \rightarrow TA_d$$

$$त + \quad ○ \quad \rightarrow त्$$

It could be assumed that a dead consonant may be combined with a vowel letter or sign to represent a CV orthographic syllable. Some non-Unicode implementations have used this approach; however, this is not done in implementations of the Unicode Standard. Instead, a CV orthographic syllable is represented with a (live) consonant followed by a dependent vowel. A dead consonant should not be followed either by an independent vowel letter or by a dependent vowel sign in an attempt to create an alternative representation of a CV orthographic syllable.

Atomic Representation of Consonant Letters. Consonant letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. In particular, consonant half forms are dead-consonant forms that often resemble a full consonant form minus a vertical stem. This vertical stem is visually similar to the vowel sign denoting /ā/, U+093E DEVANAGARI VOWEL SIGN AA. *Table 12-2* shows atomic consonant letters in Devanagari that could be graphically analyzed this way, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-2. Devanagari Atomic Consonants

For	Use	Do Not Use
ख	0916	<0916, 094D, 093E>, <0916, 094D, 200D, 093E>
ग	0917	<0917, 094D, 093E>, <0917, 094D, 200D, 093E>

घ	0918	<0918, 094D, 093E>, <0918, 094D, 200D, 093E>
च	091A	<091A, 094D, 093E>, <091A, 094D, 200D, 093E>
ज	091C	<091C, 094D, 093E>, <091C, 094D, 200D, 093E>
झ	091D	<091D, 094D, 093E>, <091D, 094D, 200D, 093E>
ञ	091E	<091E, 094D, 093E>, <091E, 094D, 200D, 093E>
ण	0923	<0923, 094D, 093E>, <0923, 094D, 200D, 093E>
त	0924	<0924, 094D, 093E>, <0924, 094D, 200D, 093E>
थ	0925	<0925, 094D, 093E>, <0925, 094D, 200D, 093E>
ध	0927	<0927, 094D, 093E>, <0927, 094D, 200D, 093E>
न	0928	<0928, 094D, 093E>, <0928, 094D, 200D, 093E>
ऩ	0929	<0929, 094D, 093E>, <0929, 094D, 200D, 093E>, <0928, 093C, 094D, 093E>, <0928, 093C, 094D, 200D, 093E>
प	092A	<092A, 094D, 093E>, <092A, 094D, 200D, 093E>
ब	092C	<092C, 094D, 093E>, <092C, 094D, 200D, 093E>
भ	092D	<092D, 094D, 093E>, <092D, 094D, 200D, 093E>
म	092E	<092E, 094D, 093E>, <092E, 094D, 200D, 093E>
य	092F	<092F, 094D, 093E>, <092F, 094D, 200D, 093E>
ल	0932	<0932, 094D, 093E>, <0932, 094D, 200D, 093E>
व	0935	<0935, 094D, 093E>, <0935, 094D, 200D, 093E>
श	0936	<0936, 094D, 093E>, <0936, 094D, 200D, 093E>
ष	0937	<0937, 094D, 093E>, <0937, 094D, 200D, 093E>
स	0938	<0938, 094D, 093E>, <0938, 094D, 200D, 093E>
ख़	0959	<0959, 094D, 093E>, <0959, 094D, 200D, 093E>, <0916, 093C, 094D, 093E>, <0916, 093C, 094D, 200D, 093E>
ग़	095A	<095A, 094D, 093E>, <095A, 094D, 200D, 093E>, <0917, 093C, 094D, 093E>, <0917, 093C, 094D, 200D, 093E>
ज़	095B	<095B, 094D, 093E>, <095B, 094D, 200D, 093E>, <091C, 093C, 094D, 093E>, <091C, 093C, 094D, 200D, 093E>
य़	095F	<095F, 094D, 093E>, <095F, 094D, 200D, 093E>, <092F, 093C, 094D, 093E>, <092F, 093C, 094D, 200D, 093E>
ज़	0979	<0979, 094D, 093E>, <0979, 094D, 200D, 093E>
ष	097A	<097A, 094D, 093E>, <097A, 094D, 200D, 093E>
ग	097B	<097B, 094D, 093E>, <097B, 094D, 200D, 093E>
ज	097C	<097C, 094D, 093E>, <097C, 094D, 200D, 093E>
ड	097E	<097E, 094D, 093E>, <097E, 094D, 200D, 093E>
ब	097F	<097F, 094D, 093E>, <097F, 094D, 200D, 093E>

The practice of using atomic consonants to represent letters is recommended. Using a half-form plus stems should be avoided.

Consonant Conjuncts. The Indic scripts are noted for a large number of consonant conjunct forms that serve as orthographic abbreviations (ligatures) of two or more adjacent letterforms. This abbreviation takes place only in the context of a *consonant cluster.* An orthographic consonant cluster is defined as a sequence of characters that represents one or more dead consonants (denoted C_d) followed by a normal, live consonant letter (denoted C_l).

Under normal circumstances, a consonant cluster is depicted with a conjunct glyph if such a glyph is available in the current font. In the absence of a conjunct glyph, the one or more dead consonants that form part of the cluster are depicted using half-form glyphs. In the absence of half-form glyphs, the dead consonants are depicted using the nominal consonant forms combined with visible virama signs (see *Figure 12-2*).

Figure 12-2. Conjunct Formations in Devanagari

(1) GA_d + DHA_l → GA_h + DHA_n (3) KA_d + SSA_l → $K.SSA_n$

ग् + ध → ग्ध   क् + ष → क्ष

(2) KA_d + KA_l → $K.KA_n$ (4) RA_d + KA_l → KA_l + RA_{sup}

क् + क → क्क   र् + क → र्क
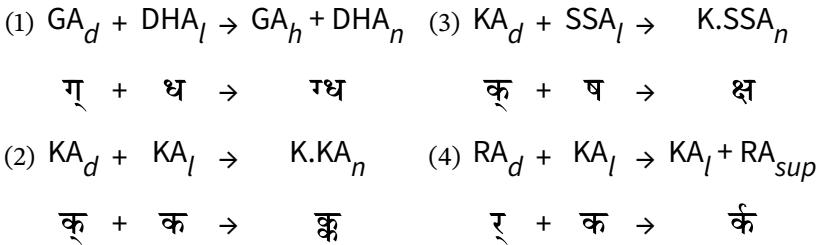
A number of types of conjunct formations appear in these examples: (1) a half-form of GA in its combination with the full form of DHA; (2) a vertical conjunct K.KA; and (3) a fully ligated conjunct K.SSA, in which the components are no longer distinct. In example (4) in *Figure 12-2*, the dead consonant RA_d is depicted with the nonspacing combining mark RA_{sup} (*repha*).

A consonant conjunct form can take a virama and so become a dead consonant conjunct form. A dead consonant conjunct form can be followed by another consonant letter, and so form a multi-consonant conjunct. For example, *Figure 12-3* illustrates a three-consonant conjunct form, P.S.YA.

Figure 12-3. Multi-Consonant Conjuncts in Devanagari

$$PA_d + SA_d + YA_l \rightarrow P.S.YA_n$$

$$प् \ + \ स् \ + \ य \ \rightarrow \ प्स्य$$

A well-designed Indic script font may contain hundreds of conjunct glyphs, but they are not encoded as Unicode characters because they are the result of ligation of distinct letters. Indic script rendering software must be able to map appropriate combinations of characters in context to the appropriate conjunct glyphs in fonts.

A dead consonant conjunct may have an appearance like a half form, because the vertical stem of the last consonant is removed. As a result, a live consonant conjunct could be analyzed visually as consisting of the dead, consonant-conjunct half form plus the vowel sign /ā/. As in the case of consonant letters, the live form should not be represented using a half form followed by U+093E DEVANAGARI VOWEL SIGN AA. *Table 12-3* shows some examples of live consonant conjuncts that exhibit this visual pattern, but that should not be represented with fully analyzed sequences. *Table 12-3* also shows the sequence of code points that should be used to represent these conjuncts in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-3. Devanagari Consonant Conjuncts

For	Use	Do Not Use
क्च	<0915, 094D, 091A>	<0915, 094D, 091A, 094D, 093E>, <0915, 094D, 091A, 094D, 200D, 093E>
क्ष	<0915, 094D, 0937>	<0915, 094D, 0937, 094D, 093E>, <0915, 094D, 0937, 094D, 200D, 093E>
त्त	<0924, 094D, 0924>	<0924, 094D, 0924, 094D, 093E>, <0924, 094D, 0924, 094D, 200D, 093E>
न्त	<0928, 094D, 0924>	<0928, 094D, 0924, 094D, 093E>, <0928, 094D, 0924, 094D, 200D, 093E>

Note that these are illustrative examples only. There are many consonant conjuncts that could be visually analyzed in the same way, and the same principle applies to all such cases: these should not be represented as dead conjunct plus vowel sign sequences. The practice of using atomic consonants to represent letters is recommended. Using a half-form plus stems should be avoided.

Explicit Virama (Halant). Normally a virama character serves to create dead consonants that are, in turn, combined with subsequent consonants to form conjuncts. This behavior usually results in a virama sign not being depicted visually. Occasionally, this default behavior is not desired when a dead consonant should be excluded from conjunct formation, in which case the virama sign is visibly rendered. To

accomplish this goal, the Unicode Standard adopts the convention of placing the character U+200C ZERO WIDTH NON-JOINER immediately after the encoded dead consonant that is to be excluded from conjunct formation. In this case, the virama sign is always depicted as appropriate for the consonant to which it is attached.

For example, in *Figure 12-4*, the use of ZERO WIDTH NON-JOINER prevents the default formation of the conjunct form क्ष ($K.SSA_n$).

Figure 12-4. Preventing Conjunct Forms in Devanagari

$$KA_d + ZWNJ + SSA_l \rightarrow KA_d + SSA_n$$

क् + [ZW NJ] + ष → क्‍ष

Explicit Half-Consonants. When a dead consonant participates in forming a conjunct, the dead consonant form is often absorbed into the conjunct form, such that it is no longer distinctly visible. In other contexts, the dead consonant may remain visible as a *half-consonant form*. In general, a half-consonant form is distinguished from the nominal consonant form by the loss of its inherent vowel stem, a vertical stem appearing to the right side of the consonant form. In other cases, the vertical stem remains but some part of its right-side geometry is missing.

In certain cases, it is desirable to prevent a dead consonant from assuming full conjunct formation yet still not appear with an explicit virama. In these cases, the half-form of the consonant is used. To explicitly encode a half-consonant form, the Unicode Standard adopts the convention of placing the character U+200D ZERO WIDTH JOINER immediately after the encoded dead consonant. The ZERO WIDTH JOINER denotes a nonvisible letter that presents linking or cursive joining behavior on either side (that is, to the previous or following letter). Therefore, in the present context, the ZERO WIDTH JOINER may be considered to present a context to which a preceding dead consonant may join so as to create the half-form of the consonant.

For example, if C_h denotes the half-form glyph of consonant C, then a half-consonant form is represented as shown in *Figure 12-5*.

Figure 12-5. Half-Consonants in Devanagari

$$KA_d + ZWJ + SSA_l \rightarrow KA_h + SSA_n$$

क् + [ZW J] + ष → क्ष

In the absence of the ZERO WIDTH JOINER, the sequence in *Figure 12-5* would normally produce the full conjunct form क्ष ($K.SSA_n$).

This encoding of half-consonant forms also applies in the absence of a base letter-form. That is, this technique may be used to encode independent half-forms, as shown in *Figure 12-6*.

Figure 12-6. Independent Half-Forms in Devanagari

$$\text{GA}_d + \text{ZWJ} \rightarrow \text{GA}_h$$

$$\text{गृ} + \boxed{\text{zwj}} \rightarrow \text{र}$$

Other Indic scripts have similar half-forms for the initial consonants of a conjunct. Some, such as Oriya, also have similar half-forms for the final consonants; those are represented as shown in *Figure 12-7*.

Figure 12-7. Half-Consonants in Oriya

$$\text{NGA}_n + \text{ZWJ} + \text{VIRAMA} + \text{KA}_l \rightarrow \text{NGA}_l + \text{KA}_h$$

$$\text{ଙ} + \boxed{\text{zwj}} + \text{ୂ} + \text{କ} \rightarrow \text{ଙ୍କ}$$

In the absence of the ZERO WIDTH JOINER, the sequence in *Figure 12-7* would normally produce the full conjunct form ଙ୍କ (NG.KA$_n$).

Consonant Forms. In summary, each consonant may be encoded such that it denotes a live consonant, a dead consonant that may be absorbed into a conjunct, the half-form of a dead consonant, or a dead consonant with an overt halant that does not get absorbed into a conjunct (see *Figure 12-8*).

Figure 12-8. Consonant Forms in Devanagari and Oriya

$$\text{क} + \text{ृ} + \text{ष} \rightarrow \text{क्ष} \qquad \text{K.SSA}_n$$

$$\text{क} + \text{ृ} + \boxed{\text{zwj}} + \text{ष} \rightarrow \text{क्ष} \qquad \text{KA}_h + \text{SSA}_n$$

$$\text{क} + \text{ृ} + \boxed{\text{znj}} + \text{ष} \rightarrow \text{क्ष} \qquad \text{KA}_d + \text{SSA}_n$$

$$\text{ଙ} + \text{ୂ} + \text{କ} \rightarrow \text{ଙ୍କ} \qquad \text{NG.KA}_n$$

$$\text{ଙ} + \boxed{\text{zwj}} + \text{ୂ} + \text{କ} \rightarrow \text{ଙ୍କ} \qquad \text{NGA}_n + \text{KA}_h$$

$$\text{ଙ} + \text{ୂ} + \boxed{\text{znj}} + \text{କ} \rightarrow \text{ଙ୍କ} \qquad \text{NGA}_d + \text{KA}_n$$

As the rendering of conjuncts and half-forms depends on the availability of glyphs in the font, the following fallback strategy should be employed:

- If the coded character sequence would normally render with a full conjunct, but such a conjunct is not available, the fallback rendering is to use half-forms. If those are not available, the fallback rendering should use an explicit (visible) virama.

- If the coded character sequence would normally render with a half-form (it contains a ZWJ), but half-forms are not available, the fallback rendering should use an explicit (visible) virama.

12.1.3 Rendering Devanagari

Rules for Rendering. This section provides more formal and detailed rules for minimal rendering of Devanagari as part of a plain text sequence. It describes the mapping between Unicode characters and the glyphs in a Devanagari font. It also describes the combining and ordering of those glyphs.

These rules provide minimal requirements for legibly rendering interchanged Devanagari text. As with any script, a more complex procedure can add rendering characteristics, depending on the font and application.

> *In a font that is capable of rendering Devanagari, the number of glyphs is greater than the number of Devanagari characters.*

Notation. In the next set of rules, the following notation applies:

C_n	Nominal glyph form of consonant C as it appears in the code charts.
C_l	A live consonant, depicted identically to C_n.
C_d	Glyph depicting the dead consonant form of consonant C.
C_h	Glyph depicting the half-consonant form of consonant C.
L_n	Nominal glyph form of a conjunct ligature consisting of two or more component consonants. A conjunct ligature composed of two consonants X and Y is also denoted $X.Y_n$.
RA_{sup}	A nonspacing combining mark glyph form of U+0930 DEVANAGARI LETTER RA positioned above or attached to the upper part of a base glyph form. This form is also known as *repha*.
RA_{sub}	A nonspacing combining mark glyph form of U+0930 DEVANAGARI LETTER RA positioned below or attached to the lower part of a base glyph form.

V_{vs} Glyph depicting the dependent vowel sign form of a vowel V.

$VIRAMA_n$ The nominal glyph form of the nonspacing combining mark
         depicting U+094D DEVANAGARI SIGN VIRAMA.

A virama character is not always depicted. When it is depicted, it adopts this nonspacing mark form.

Dead Consonant Rule. The following rule logically precedes the application of any other rule to form a dead consonant. Once formed, a dead consonant may be subject to other rules described next.

R1 *When a consonant* C_n *precedes a* $VIRAMA_n$*, it is considered to be a dead consonant* C_d*. A consonant* C_n *that does not precede* $VIRAMA_n$ *is considered to be a live consonant* C_l*.*

$$TA_n + VIRAMA_n \rightarrow TA_d$$

$$त + \quad ़ \quad \rightarrow त्$$

Consonant RA Rules. The character U+0930 DEVANAGARI LETTER RA takes one of a number of visual forms depending on its context in a consonant cluster. By default, this letter is depicted with its nominal glyph form (as shown in the code charts). In some contexts, it is depicted using one of two nonspacing glyph forms that combine with a base letterform.

R2 *If the dead consonant* RA_d *precedes a consonant, then it is replaced by the superscript nonspacing mark* RA_{sup} *, which is positioned so that it applies to the logically subsequent element in the memory representation.*

$$RA_d + KA_l \rightarrow KA_l + RA_{sup} \qquad \text{\textit{Displayed Output}}$$

$$र + क \rightarrow क + ़ \rightarrow क॔$$

$$RA_d^1 + RA_d^2 \rightarrow RA_d^2 + RA_{sup}^1$$

$$र + र \rightarrow र + ़ \rightarrow र॔$$

R3 *If the superscript mark* RA_{sup} *is to be applied to a dead consonant and that dead consonant is combined with another consonant to form a*

conjunct ligature, then the mark is positioned so that it applies to the conjunct ligature form as a whole.

$$\text{RA}_d + \text{JA}_d + \text{NYA}_l \rightarrow \text{J.NYA}_n + \text{RA}_{sup} \qquad \text{\textit{Displayed Output}}$$

र_ + ज़_ + ञ → ज़ + ॑ → र्ज़ॉ

R4 *If the superscript mark* RA_{sup} *is to be applied to a dead consonant that is subsequently replaced by its half-consonant form, then the mark is positioned so that it applies to the form that serves as the base of the consonant cluster.*

$$\text{RA}_d + \text{GA}_d + \text{GHA}_l \rightarrow \text{GA}_h + \text{GHA}_l + \text{RA}_{sup} \qquad \text{\textit{Displayed Output}}$$

र_ + ग़_ + घ → ग़ + घ + ॑ → र्घ

R5 *In conformance with the ISCII standard, the half-consonant form* RRA_h *is represented as eyelash-*RA*. This form of* RA *is commonly used in writing Marathi and Newari.*

$$\text{RRA}_n + \text{VIRAMA}_n \rightarrow \text{RRA}_h$$

ऱ + ॖ → ऱ्

R5a *For compatibility with The Unicode Standard, Version 2.0, if the dead consonant* RA_d *precedes* ZERO WIDTH JOINER*, then the half-consonant form* RA_h *, depicted as eyelash-*RA*, is used instead of* RA_{sup} *.*

$$\text{RA}_d + \text{ZWJ} \rightarrow \text{RA}_h$$

र_ + [ZWJ] → ऱ्

R6 *Except for the dead consonant* RA_d *, when a dead consonant* C_d *precedes the live consonant* RA_l, *then* C_d *is replaced with its nominal form* C_n *, and* RA *is replaced by the subscript nonspacing mark* RA_{sub}, *which is positioned so that it applies to* C_n.

$$\text{TTHA}_d + \text{RA}_l \rightarrow \text{TTHA}_n + \text{RA}_{sub} \qquad \textit{Displayed Output}$$

ठ़ + र → ठ + ् र → ठ्र

R7 *For certain consonants, the mark* RA$_{sub}$ *may graphically combine with the consonant to form a conjunct ligature form. These combinations, such as the one shown here, are further addressed by the ligature rules described shortly.*

$$\text{PHA}_d + \text{RA}_l \rightarrow \text{PHA}_n + \text{RA}_{sub} \qquad \textit{Displayed Output}$$

फ़ + र → फ + ्र → फ्र

R8 *If a dead consonant (other than* RA$_d$ *) precedes* RA$_d$ *, then the substitution of* RA *for* RA$_{sub}$ *is performed as described above; however, the* VIRAMA *that formed* RA$_d$ *remains so as to form a dead consonant conjunct form.*

$$\text{TA}_d + \text{RA}_d \rightarrow \text{TA}_n + \text{RA}_{sub} + \text{VIRAMA}_n \rightarrow \text{T.RA}_d$$

त़ + ऱ → त + ्र + ्र → त्ऱ

A dead consonant conjunct form that contains an absorbed RA$_d$ *may subsequently combine to form a multipart conjunct form.*

$$\text{T.RA}_d + \text{YA}_l \rightarrow \text{T.R.YA}_n$$

त्ऱ + य → त्र्य

Modifier Mark Rules. In addition to vowel signs, three other types of combining marks may be applied to a component of an orthographic syllable or to the syllable as a whole: *nukta*, *bindus*, and *svaras* (such as U+0951 DEVANAGARI STRESS SIGN UDATTA and U+0952 DEVANAGARI STRESS SIGN ANUDATTA).

R9 *The nukta sign, which modifies a consonant form, is placed immediately after the consonant in the memory representation and is attached to that consonant in rendering. If the consonant represents a dead consonant, then* NUKTA *should precede* VIRAMA *in the memory representation.*

$$KA_n + NUKTA_n + VIRAMA_n \rightarrow QA_d$$

क + ़ + ् → क़्

R10 *Other modifying marks, in particular bindus and svaras, apply to the orthographic syllable as a whole and should follow (in the memory representation) all other characters that constitute the syllable. The bindus should follow any vowel signs, and the svaras should come last. A bindu and svara are placed side by side when they coexist on top of an orthographic syllable; the horizontal order may vary according to typographic concerns.*

$$KA_n + AA_{vs} + CANDRABINDU_n$$

क + ा + ँ → काँ

Ligature Rules. Subsequent to the application of the rules just described, a set of rules governing ligature formation apply. The precise application of these rules depends on the availability of glyphs in the current font being used to display the text.

R11 *If a dead consonant immediately precedes another dead consonant or a live consonant, then the first dead consonant may join the subsequent element to form a two-part conjunct ligature form.*

$$JA_d + NYA_l \rightarrow J.NYA_n \qquad\qquad TTA_d + TTHA_l \rightarrow TT.TTHA_n$$

ज् + अ → ज्ञ ट् + ठ → ट्ठ

R12 *A conjunct ligature form can itself behave as a dead consonant and enter into further, more complex ligatures.*

$$SA_d + TA_d + RA_n \rightarrow SA_d + T.RA_n \rightarrow S.T.RA_n$$

स् + त् + र → स् + त्र → स्त्र

A conjunct ligature form can also produce a half-form.

$$K.SSA_d + YA_l \rightarrow K.SS_h + YA_n$$

क्ष् + य → क्ष्य

R13 *If a nominal consonant or conjunct ligature form precedes* RA_{sub} *as a result of the application of rule R6, then the consonant or ligature form may join with* RA_{sub} *to form a multipart conjunct ligature (see rule R6 for more information).*

$$KA_n + RA_{sub} \rightarrow K.RA_n \qquad\qquad PHA_n + RA_{sub} \rightarrow PH.RA_n$$

$$\text{क} + \text{ } \text{ } \rightarrow \text{क्र} \qquad\qquad\qquad \text{फ} + \text{ } \text{ } \rightarrow \text{फ्र}$$

R14 *In some cases, other combining marks will combine with a base consonant, either attaching at a nonstandard location or changing shape. In minimal rendering, there are only two cases:* RA_l *with* U_{vs} *or* UU_{vs}.

$$RA_l + U_{vs} \rightarrow RU_n \qquad\qquad RA_l + UU_{vs} \rightarrow RUU_n$$

$$\text{र} + \text{ } \text{ } \rightarrow \text{रु} \qquad\qquad\quad \text{र} + \text{ } \text{ } \rightarrow \text{रू}$$

Memory Representation and Rendering Order. The storage of plain text in Devanagari and all other Indic scripts generally follows phonetic order; that is, a CV syllable with a dependent vowel is always encoded as a consonant letter C followed by a vowel sign V in the memory representation. This order is employed by the ISCII standard and corresponds to both the phonetic order and the keying order of textual data (see *Figure 12-9*).

Figure 12-9. Rendering Order in Devanagari

Character Order   Glyph Order

$$KA_n + I_{vs} \rightarrow I_{vs} + KA_n$$

$$\text{क} + \text{ि} \rightarrow \text{कि}$$

Because Devanagari and other Indic scripts have some dependent vowels that must be depicted to the left side of their consonant letter, the software that renders the Indic scripts must be able to reorder elements in mapping from the logical (character) store to the presentational (glyph) rendering. For example, if C_n denotes the nominal form of consonant C, and V_{vs} denotes a left-side dependent vowel sign form of vowel V, then a reordering of glyphs with respect to encoded characters occurs as just shown.

R15 **When the dependent vowel I$_{vs}$ is used to override the inherent vowel of a syllable, it is always written to the extreme left of the orthographic syllable. If the orthographic syllable contains a consonant cluster, then this vowel is always depicted to the left of that cluster.**

$$TA_d + RA_l + I_{vs} \rightarrow T.RA_n + I_{vs} \rightarrow I_{vs} + T.RA_n$$

$$त् \;+\; र \;+\; ि \rightarrow \;\;\; त्र \;\;+\; ि \rightarrow \;\;\;\;\; त्रि$$

R16 **The presence of an explicit virama (either caused by a ZWNJ or by the absence of a conjunct in the font) blocks this reordering, and the dependent vowel I$_{vs}$ is rendered after the rightmost such explicit virama.**

$$TA_d + ZWNJ + RA_l + I_{vs} \rightarrow TA_d + I_{vs} + RA_l$$

$$त् \;+\; \boxed{\tiny ZWNJ} \;+\; र \;+\; ि \rightarrow \;\;\;\;\; त्रि$$

Alternative Forms of Cluster-Initial RA. In addition to *reph* (rule R2) and *eyelash* (rule R5a), a cluster-initial RA may also take its nominal form while the following consonant takes a reduced form. This behavior is required by languages that make a morphological distinction between "*reph* on YA" and "RA with reduced YA", such as Braj Bhasha. To trigger this behavior, a ZWJ is placed immediately before the *virama* to request a reduced form of the following consonant, while preventing the formation of *reph*, as shown in the third example below.

$$र + \;\;\;\; ्\;\;\;\; + य \rightarrow र्य$$
$$र + ् + \boxed{\tiny ZWJ} + य \rightarrow र्य$$
$$र + \boxed{\tiny ZWJ} + ् + य \rightarrow र्य$$

Similar, special rendering behavior of cluster-initial RA is noted in other scripts of India. See, for example, "Interaction of Repha and Ya-phalaa" in *Section 12.2, Bengali (Bangla)*, "Reph" in *Section 12.7, Telugu*, and "Consonant Clusters Involving RA" in *Section 12.8, Kannada*.

Sample Half-Forms. *Table 12-4* shows examples of half-consonant forms that are commonly used with the Devanagari script. These forms are glyphs, not characters. They may be encoded explicitly using ZERO WIDTH JOINER as shown. In normal conjunct formation, they may be used spontaneously to depict a dead consonant in combination with subsequent consonant forms.

Table 12-4. Sample Devanagari Half-Forms

क	+	९	+	ZW J	→	क्	न	+	९	+	ZW J	→	न्

क + ९ + ZW J → क्   न + ९ + ZW J → न्

ख + ९ + ZW J → ख्   प + ९ + ZW J → प्

ग + ९ + ZW J → ग्   फ + ९ + ZW J → फ्

घ + ९ + ZW J → घ्   ब + ९ + ZW J → ब्

च + ९ + ZW J → च्   भ + ९ + ZW J → भ्

ज + ९ + ZW J → ज्   म + ९ + ZW J → म्

झ + ९ + ZW J → झ्   य + ९ + ZW J → य्

अ + ९ + ZW J → ञ्   ल + ९ + ZW J → ल्

ण + ९ + ZW J → ण्   व + ९ + ZW J → व्

त + ९ + ZW J → त्   श + ९ + ZW J → श्

थ + ९ + ZW J → थ्   ष + ९ + ZW J → ष्

ध + ९ + ZW J → ध्   स + ९ + ZW J → स्

Sample Ligatures. *Table 12-5* shows examples of conjunct ligature forms that are commonly used with the Devanagari script. These forms are glyphs, not characters. Not every writing system that employs this script uses all of these forms; in particular, many of these forms are used only in writing Sanskrit texts. Furthermore, individual fonts may provide fewer or more ligature forms than are depicted here.

Table 12-5. Sample Devanagari Ligatures

क + ् + क → क्क	ट + ् + ठ → ट्ठ
क + ् + त → क्त	ठ + ् + ठ → ठ्ठ
क + ् + र → क्र	ड + ् + ग → ड्ग
क + ् + ष → क्ष	ड + ् + ड → ड्ड
ङ + ् + क → ङ्क	ड + ् + ढ → ड्ढ
ङ + ् + ख → ङ्ख	त + ् + त → त्त
ङ + ् + ग → ङ्ग	त + ् + र → त्र
ङ + ् + घ → ङ्घ	न + ् + न → न्न
ञ + ् + ज → ञ्ज	फ + ् + र → फ्र
ज + ् + ञ → ज्ञ	श + ् + र → श्र
द + ् + घ → द्घ	ह + ् + म → ह्म
द + ् + द → द्द	ह + ् + य → ह्य
द + ् + ध → द्ध	ह + ् + ल → ह्ल
द + ् + ब → द्ब	ह + ् + व → ह्व
द + ् + भ → द्भ	ह + ् → हृ
द + ् + म → द्म	र + ु → रु
द + ् + य → द्य	र + ू → रू
द + ् + व → द्व	स + ् + त्र → स्त्र
ट + ् + ट → ट्ट	

Ligature Forms for Ra + Vocalic Liquids. The phonological sequence /r vocalic_r/, expressed with the character sequence <U+0930 ra, U+0943 vocalic_r>, can graphically appear as either of two forms, as shown in the first row of . It may appear as the full independent vowel form of the vocalic_r, with a superscript *repha*

form of the ra (V + RA$_{sup}$): ऋ. Alternatively, it may appear as the full letter form of the ra with the subscript, dependent form of the vocalic_r (RA$_n$ + V$_{vs}$): र‍ृ. Similarly, the phonological sequences with the other vocalic sounds (*rr, l, ll*) have two written forms, as shown in *Table 12-6*.

Table 12-6. RA + Vocalic Letter Ligature Forms

र + ◌ृ → ऋ or र‍ृ
र + ◌ॄ → ॠ or र‍ॄ
र + ◌ॢ → ऌ or र‍ॢ
र + ◌ॣ → ॡ or र‍ॣ

The graphical forms displayed above with the reph (RA$_{sup}$) should not be represented by sequences of RA + virama + independent vowel, as such sequences violate the general encoding principles of the script. CV orthographic syllables are not represented by consonant + virama + independent vowel.

The practice of writing these phonological sequences as a *reph* on an independent vocalic liquid letter is also observed in other Indic scripts, such as Gujarati, Oriya, Telugu, Kannada, and Bhaiksuki.

Sample Half-Ligature Forms. In addition to half-form glyphs of individual consonants, half-forms are used to depict conjunct ligature forms. A sample of such forms is shown in *Table 12-7*. These forms are glyphs, not characters. They may be encoded explicitly using ZERO WIDTH JOINER as shown. In normal conjunct formation, they may be used spontaneously to depict a conjunct ligature in combination with subsequent consonant forms.

Table 12-7. Sample Devanagari Half-Ligature Forms

क + ◌् + ष + ◌् + [ZWJ] → क्ष
ज + ◌् + ञ + ◌् + [ZWJ] → ज्ञ
त + ◌् + त + ◌् + [ZWJ] → त्त
त + ◌् + र + ◌् + [ZWJ] → त्र
श + ◌् + र + ◌् + [ZWJ] → श्र

Language-Specific Allographs. In Marathi, Nepali, and some South Indian orthographies, variant glyphs are preferred for certain letters and digits. These include U+091D DEVANAGARI LETTER JHA, U+0932 DEVANAGARI LETTER LA, U+0936 DEVANAGARI LETTER SHA, and the digits five, eight, and nine, as shown in *Table 12-8*. Marathi also makes use of the "eyelash" form of the letter RA, as discussed in rule R5.

Table 12-8. Marathi and Nepali Allographs

Code Point	Hindi	Marathi	Nepali
U+091D JHA	झ	झ	भ
U+0932 LA	ल	ल	ल
U+0936 SHA	श	श	श
U+096B FIVE	५	५	५
U+096E EIGHT	८	८	८
U+096F NINE	९	९	९

In addition, various languages written in Devanagari (or sometimes their various orthographic traditions) tend to have different preferences for formation of certain ligatures (see the text on "Sample Ligatures," earlier in this section). For example, modern Nepali orthographies prefer a smaller number of ligatures than commonly used in Hindi or Marathi.

Combining Marks. Devanagari and other Indic scripts have a number of combining marks that could be considered diacritic. One class of these marks, known as bindus, is represented by U+0901 DEVANAGARI SIGN CANDRABINDU and U+0902 DEVANAGARI SIGN ANUSVARA. These marks indicate nasalization or final nasal closure of a syllable. U+093C DEVANAGARI SIGN NUKTA is a true diacritic. It is used to extend the basic set of consonant letters by modifying them (with a subscript dot in Devanagari) to create new letters.

U+0951 DEVANAGARI STRESS SIGN UDATTA and U+0952 DEVANAGARI STRESS SIGN ANUDATTA are tone marks used in the representation of Vedic text in Devanagari. These two combining marks may also occur in the representation of Vedic texts written in other scripts, including transliterations in the Latin script. They are given the Indic_Syllabic_Category value of Cantillation_Mark.

U+0953 DEVANAGARI GRAVE ACCENT and U+0954 DEVANAGARI ACUTE ACCENT were originally encoded for Latin transliteration of Sanskrit text. However, such use is now discouraged, and Latin transliterations should simply use the generic combining marks, U+0300 COMBINING GRAVE ACCENT and U+0301 COMBINING ACUTE ACCENT. Because U+0953 and U+0954 are not intended to be used with the Devanagari script, they have no explicit property values for Indic_Positional_Category and Indic_Syllabic_Category.

12.1.4 Devanagari Digits, Punctuation, and Symbols

Digits. Each Indic script has a distinct set of digits appropriate to that script. These digits may or may not be used in ordinary text in that script. European digits have displaced the Indic script forms in modern usage in many of the scripts. Some Indic scripts—notably Tamil—lacked a distinct digit for zero in their traditional numerical systems, but adopted a zero based on general Indian practice.

Punctuation. U+0964 । DEVANAGARI DANDA is similar to a full stop. U+0965 ॥ DEVANAGARI DOUBLE DANDA marks the end of a verse in traditional texts. The term *danda* is from Sanskrit, and the punctuation mark is generally referred to as a *viram* instead in Hindi. Although the *danda* and *double danda* are encoded in the Devanagari block, the intent is that they be used as common punctuation for all the major scripts of India covered by this chapter. *Danda* and *double danda* punctuation marks are not separately encoded for some Indic scripts, such as Gujarati, Gurmukhi, and Oriya. However, analogous punctuation marks for other Brahmi-derived scripts *are* separately encoded, particularly for scripts used primarily outside of India.

Many modern languages written in the Devanagari script intersperse punctuation derived from the Latin script. Thus U+002C COMMA and U+002E FULL STOP are freely used in writing Hindi, and the *danda* is usually restricted to more traditional texts. However, the *danda* may be preserved when such traditional texts are transliterated into the Latin script.

Other Symbols. U+0970 ॰ DEVANAGARI ABBREVIATION SIGN appears after letters or combinations of letters and marks the sequence as an abbreviation. It is intended specifically for Devanagari script-based abbreviations, such as the Devanagari rupee sign. Other symbols and signs most commonly occurring in Vedic texts are encoded in the Devanagari Extended and Vedic Extensions blocks and are discussed in the text that follows.

The *svasti* (or well-being) signs often associated with the Hindu, Buddhist, and Jain traditions are encoded in the Tibetan block. See *Section 13.4, Tibetan* for further information.

12.1.5 Extensions in the Main Devanagari Block

Sindhi Letters. The characters U+097B DEVANAGARI LETTER GGA, U+097C DEVANAGARI LETTER JJA, U+097E DEVANAGARI LETTER DDDA, and U+097F DEVANAGARI LETTER BBA are used to write Sindhi implosive consonants. Previous versions of the Unicode Standard recommended representing those characters as a combination of the usual consonants with *nukta* and *anudātta*, but those combinations are no longer recommended.

Konkani. Konkani makes use of additional sounds that can be represented with combinations such as U+091A devanagari letter ca plus U+093C devanagari sign nukta and U+091F devanagari letter tta plus U+0949 devanagari vowel sign candra o.

Kashmiri Letters. There are several letters for use with Kashmiri when written in Devanagari script. Long and short versions of the independent vowel letters are encoded in the range U+0973..U+0977. The corresponding dependent vowel signs are U+093A devanagari vowel sign oe, U+093B devanagari vowel sign ooe, and U+094F devanagari vowel sign aw. The forms of the independent vowels for Kashmiri are constructed by using the glyphs of the matras U+093B devanagari vowel sign ooe, U+094F devanagari vowel sign aw, U+0956 devanagari vowel sign ue, and U+0957 devanagari vowel sign uue as diacritics on U+0905 devanagari letter a. However, for representation of independent vowels in Kashmiri, use the encoded, composite characters in the range U+0973..U+0977 and not the visually equivalent sequences of U+0905 devanagari letter a plus the matras. See *Table 12-1*. A few of the letters identified as being used for Kashmiri are also used to write the Bihari languages.

Bodo, Dogri, and Maithili. The orthographies of the Bodo, Dogri, and Maithili languages of India make use of U+02BC " ' " modifier letter apostrophe, either as a tone mark or as a length mark. In Bodo and Dogri, this character functions as a tone mark, called *gojau kamaa* in Bodo and *sur chinha* in Dogri. In Dogri, the tone mark occurs after short vowels, including inherent vowels, and indicates a high-falling tone. After Dogri long vowels, a high-falling tone is written instead using U+0939 devanagari letter ha.

In Maithili, U+02BC " ' " modifier letter apostrophe is used to indicate the prolongation of a short *a* and to indicate the truncation of words. This sign is called *bikari kaamaa*.

Examples illustrating the use of U+02BC " ' " modifier letter apostrophe in Bodo, Dogri, and Maithili are shown in *Figure 12-10*. The Maithili examples show the same sentence, first in full form, and then using U+02BC to show truncation of words.

Figure 12-10. Use of Apostrophe in Bodo, Dogri and Maithili

Language	Examples	Meaning
Bodo	खर' दख'ना	head type of Bodo dress
Dogri	ख'ल्ल ति'लकना	down to slip
Maithili	कतए पड़ाए गेलह? कत' पड़ा' गेल'?	} Where did you go away?

In both Dogri and Maithili, an *avagraha sign*, U+093D DEVANAGARI SIGN AVAGRAHA, is used to indicate extra-long vowels. An example of the contrastive use of this *avagraha sign* is shown for Dogri in *Figure 12-11*.

Figure 12-11. Use of Avagraha in Dogri

Example	Meaning
तला	sole
तलाऽ	pond

Letters for Bihari Languages. A number of the Devanagari vowel letters have been used to write the Bihari languages Bhojpuri, Magadhi, and Maithili, as listed in *Table 12-9*.

Table 12-9. Devanagari Vowels Used in Bihari Languages

U+090E ऎ DEVANAGARI LETTER SHORT E
U+0912 ऒ DEVANAGARI LETTER SHORT O
U+0946 ॆ DEVANAGARI VOWEL SIGN SHORT E
U+094A ॊ DEVANAGARI VOWEL SIGN SHORT O
U+0973 ऒ DEVANAGARI LETTER OE
U+0974 ऒ DEVANAGARI LETTER OOE
U+0975 औ DEVANAGARI LETTER AW
U+093A � DEVANAGARI VOWEL SIGN OE
U+093B ꣰ DEVANAGARI VOWEL SIGN OOE
U+094F ॏ DEVANAGARI VOWEL SIGN AW

Letter Short a. The character U+0904 DEVANAGARI LETTER SHORT A is used to denote a short *e* in the Awadi language, an Indo-Aryan language spoken in the north Indian state of Uttar Pradesh and southern Nepal. A publisher in Lucknow, Uttar Pradesh also uses it in Hindi translations and Devanagari transliterations of the Kannada, Telugu, Tamil, Malayalam and Kashmiri languages.

Prishthamatra Orthography. In the historic Prishthamatra orthography, the vowel signs for *e*, *ai*, *o*, and *au* are represented using U+094E DEVANAGARI VOWEL SIGN PRISHTHAMATRA E (which goes on the left side of the consonant) alone or in combination with one of U+0947 DEVANAGARI VOWEL SIGN E, U+093E DEVANAGARI VOWEL SIGN AA or U+094B DEVANAGARI VOWEL SIGN O. *Table 12-10* shows those combinations applied to *ka*. In the underlying representation of text, U+094E should be first in the sequence of dependent vowel signs after the consonant, and may be followed by U+0947, U+093E or U+094B.

Table 12-10. Prishthamatra Orthography

	Prishthamatra Orthography	Modern Orthography
ke	कॆ <0915, 094E>	के <0915, 0947>
kai	कॆ <0915, 094E, 0947>	कै <0915, 0948>
ko	काॆ <0915, 094E, 093E>	को <0915, 094B>
kau	कॊ <0915, 094E, 094B>	कौ <0915, 094C>

12.1.6 Devanagari Extended: U+A8E0–U+A8FF

This block of characters is used chiefly for Vedic Sanskrit, although many of the characters are generic and can be used by other Indic scripts. The block includes a set of combining digits, letters, and *avagraha* which is used as a system of cantillation marks in the early Vedic Sanskrit texts. The Devanagari Extended block also includes nasalization marks (*candrabindu*), and a number of editorial marks.

The Devanagari Extended block, as well as the Vedic Extensions block and the Devanagari block, include characters that are used to indicate tone in Vedic Sanskrit. Indian linguists describe tone as a feature of vowels, shared by the consonants in the same syllable, or as a feature of syllables. In Vedic, vowels are marked for tone, as are certain non-vocalic characters that are syllabified in Vedic recitation (*visarga* and *anusvāra*); the tone marks directly follow the vowel or other character that they modify. Vowels are categorized according to tone as either *udātta* (high-toned or "acute"), *anudātta* (low-toned or "non-acute"), *svarita* ("modulated" or dropping from high to low tone) or *ekaśruti* (monotone). Some of the symbols used for marking tone indicate different tones in different traditions. *Visarga* may be marked for all three tones. The tone marks also can indicate other modifications of vocal text, such as vibration, lengthening a vowel, or skipping a tone in a descending scale.

Cantillation marks are used to indicate length, tone, and other features in the recited text of *Sāmaveda*, and in the Kauthuma and Rāṇāyanīya traditions of *Sāmagāna*. These marks are encoded as a series of combining digits, alphabetic characters, and *avagraha* in the range U+A8E0..U+A8F1.

Cantillation Marks for the Sāmaveda. One of the four major Vedic texts is *Sāmaveda*. The text is both recited (*Sāmaveda-Saṁhitā*) and sung (*Sāmagāna*), and is marked differently for the purposes of each. Cantillation marks are used to indicate length, tone, and other features in the recited text of *Sāmaveda*, and in the Kauthuma and Rāṇāyanīya traditions of *Sāmagāna*. These marks are encoded as a series of combining digits, alphabetic characters, and *avagraha* in the range U+A8E0..U+A8F1. The marks are rendered directly over the base letter. They are represented in text immediately after the syllable they modify.

In certain cases, two marks may occur over a letter: U+A8E3 COMBINING DEVANA-GARI DIGIT THREE may be followed by U+A8EC COMBINING DEVANAGARI LETTER KA, for example. Although no use of U+A8E8 COMBINING DEVANAGARI DIGIT EIGHT has been found in the *Sāmagāna*, it is included to provide a complete set of 0–9 digits. The combining marks encoded for the *Sāmaveda* do not include characters that may appear as subscripts and superscripts in the Jaiminīya tradition of *Sāmagāna*, which used interlinear annotation. Interlinear annotation may be rendered using Ruby and may be represented by means of markup or other higher-level protocols.

Nasalization Marks. The set of spacing marks in the range U+A8F2..U+A8F7 include the term *candrabindu* in their names and indicate nasalization. These marks are all aligned with the headline. Note that U+A8F2 DEVANAGARI SIGN SPACING CANDRABINDU is lower than the U+0901 DEVANAGARI SIGN CANDRABINDU.

Editorial Marks. A set of editorial marks is encoded in the range U+A8F8..U+A8FB for use with Devanagari. U+A8F9 DEVANAGARI GAP FILLER signifies an intentional gap that would ordinarily be filled with text. In contrast, U+A8FB DEVANAGARI HEADSTROKE indicates illegible gaps in the original text. The glyph for DEVANAGARI HEADSTROKE should be designed so that it does not connect to the headstroke of the letters beside it, which will make it possible to indicate the number of illegible sylla-bles in a given space. U+A8F8 DEVANAGARI SIGN PUSHPIKA acts as a filler in text, and is commonly flanked by double dandas. U+A8FA DEVANAGARI CARET, a zero-width spacing character, marks the insertion point of omitted text, and is placed at the insertion point between two orthographic syllables. It can also be used to indicate word division.

12.1.7 Devanagari Extended-A: U+11B00–U+11B5F

Bhale Mīṇḍu. Characters in the range of U+11B00..U+11B4F represent auspicious signs used in benedictions of Jaina manuscripts and inscriptions in western and central India. They are functionally similar to, but distinct from *siddham* signs such as U+A8FC DEVANAGARI SIGN SIDDHAM.

These auspicious signs are typically represented as sequences of up to three charac-ters: a head-mark (U+11B00, U+11B01), followed by an initial or *bhale* (U+11B02..U+11B06), and a terminal or *mīṇḍu* (U+11B09, U+0966). The sequence is usually followed by a *double danda*.

12.1.8 Vedic Extensions: U+1CD0–U+1CFF

The Vedic Extensions block includes characters that are used in Vedic texts; they may be used with Devanagari, as well as many other Indic scripts. This block includes a set of characters designating tone, grouped by the various Vedic traditions in which they occur. Characters indicating tone marks directly follow the character they

modify. Most of these marks indicate the tone of vowels, but three of them specifically indicate the tone of *visarga*.

A number of marks for nasalization are also included in the block. U+1CD3 VEDIC SIGN NIHSHVASA is a breaking mark which separates sections of Samavedic singing between which a pause is disallowed. The block also contains several Vedic signs for *ardhavisarga*, *jihvamuliya*, *upadhmaniya* and *atikrama*.

Tone Marks. The Vedic tone marks are all combining marks. The tone marks are grouped together in the code charts based upon the tradition in which they appear: they are used in the four core texts of the Vedas (*Sāmaveda*, *Yajurveda*, *Rigveda*, and *Atharvaveda*) and in the prose text on Vedic ritual (*Śatapathabrāhmaṇa*). The character U+1CD8 VEDIC TONE CANDRA BELOW is also used to identify the short vowels *e* and *o*. In this usage, the prescribed order is the Indic syllable (*aksara*), followed by U+1CD8 VEDIC TONE CANDRA BELOW and the tone mark (*svara*). When a tone mark is placed below, it appears below the VEDIC TONE CANDRA BELOW.

In addition to the marks encoded in this block, Vedic texts may use other nonspacing marks from the General Diacritics block and other blocks. For example, U+20F0 COMBINING ASTERISK ABOVE would be used to represent a mark of that shape above a Vedic letter.

Diacritics for the Visarga. A set of combining marks that serve as diacritics for the *visarga* is encoded in the range U+1CE2..U+1CE8. These marks indicate that the *visarga* has a particular tone. For example, the combination U+0903 DEVANAGARI SIGN VISARGA plus U+1CE2 VEDIC SIGN VISARGA SVARITA represents a *svarita visarga*. The upward-shaped diacritic is used for the *udātta* (high-toned), the downward-shaped diacritic for *anudātta* (low-toned), and the midline glyph indicates the *svarita* (modulated tone).

In Vedic manuscripts the tonal mark (that is, the horizontal bar, upward curve and downward curve) appears in colored ink, while the two dots of the *visarga* appear in black ink. The characters for accents can be represented using separate characters, to make it easier for color information to be maintained by means of markup or other higher-level protocols.

Nasalization Marks. A set of spacing marks and one combining mark, U+1CED VEDIC SIGN TIRYAK, are encoded in the range U+1CE9..U+1CF1. They describe phonetic distinctions in the articulation of nasals. The *gomukha* characters from U+1CE9..U+1CEC may be combined with U+0902 DEVANAGARI SIGN ANUSVARA or U+0901 DEVANAGARI SIGN CANDRABINDU. U+1CF1 VEDIC SIGN ANUSVARA UBHAYATO MUKHA may indicate a *visarga* with a tonal mark as well as a nasal. The three characters, U+1CEE VEDIC SIGN HEXIFORM LONG ANUSVARA, U+1CEF VEDIC SIGN LONG ANUSVARA, and U+1CF0 VEDIC SIGN RTHANG LONG ANUSVARA, are all synonymous and indicate a long *anusvāra* after a short vowel. U+1CED VEDIC SIGN

TIRYAK is the only combining character in this set of nasalization marks. While it appears similar to the U+094D DEVANAGARI SIGN VIRAMA, it is used to render glyph variants of nasal marks that occur in manuscripts and printed texts.

Ardhavisarga. U+1CF2 VEDIC SIGN ARDHAVISARGA is a character that marks either the *jihvāmūlīya*, a velar fricative occurring only before the unvoiced velar stops *ka* and *kha*, or the *upadhmānīya*, a bilabial fricative occurring only before the unvoiced labial stops *pa* and *pha*. *Ardhavisarga* is a spacing character. It is represented in text in visual order before the consonant it modifies.

12.2 Bengali (Bangla)

12.2.1 Bengali: U+0980–U+09FF

The term *Bengali* is used in the Unicode Standard for the script and character names. However, users of the script in the Indian state of West Bengal and the People's Republic of Bangladesh prefer *Bangla*, so the term Bangla is used in this section and elsewhere in this chapter. The Bangla script is used for writing languages such as Bangla, Assamese, Bishnupriya Manipuri, Daphla, Garo, Hallam, Khasi, Mizo, Munda, Naga, Rian, and Santali. Although the Assamese language has been written historically using regional scripts, known generally as "Kamrupi," its modern writing system is similar to that presently used for Bangla, with the addition of extra characters. The Bangla block supports the modern Assamese orthography. In the Indian state of Assam, the script is called *Asamiya* or *Assamese*.

The Bangla script is a North Indian script historically related to Devanagari.

Virama (Hasant). The Bangla script uses the Unicode virama model to form conjunct consonants. In Bangla, the virama is known as *hasant*.

Vowel Letters. Vowel letters of Indic scripts are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-11* shows the Bangla vowel letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-11. Bangla Vowel Letters

For	Use	Do Not Use
আ	0986	<0985, 09BE>
ঋ	09E0	<098B, 09C3>
ৡ	09E1	<098C, 09E2>

There is an exception to this general pattern for the representation of Bangla independent vowel letters, for the Bangla script orthography of Kokborok, a major language of Tripura state in Northeast India. Kokborok has diphthongs which can occur as initial letters. To reflect existing practice, these diphthongs are represented with two character sequences, rather than as atomic characters, as shown in *Table 12-12*. Rendering systems which support display of the Kokborok orthography need to be aware of these exceptional sequences. The sequence for *vowel letter aw* uses U+09D7 ৗ BENGALI AU LENGTH MARK, also noted in the following discussion of two-part vowel signs.

Table 12-12. Diphthong Vowel Letters in Kokborok

For	Use	Description
অ◌	<0985, 09D7>	*vowel letter aw*
উ◌া	<0989, 09BE>	*vowel letter ua*

Two-Part Vowel Signs. The Bangla script, along with a number of other Indic scripts, makes use of two-part dependent vowel signs. In these dependent vowels (*matras*) one-half of the vowel is displayed on each side of a consonant letter or cluster—for example, U+09CB ে◌া BENGALI VOWEL SIGN O and U+09CC ে◌ৗ BENGALI VOWEL SIGN AU. To provide compatibility with existing implementations of the scripts that use two-part vowel signs, the Unicode Standard explicitly encodes the right half of these vowel signs. For example, U+09D7 ◌ৗ BENGALI AU LENGTH MARK represents the right-half glyph component of U+09CC ে◌ৗ BENGALI VOWEL SIGN AU. In Bangla orthography, the *au length mark* is always used in conjunction with the left part and does not have a meaning on its own.

Special Characters. U+09F2..U+09F9 are a series of Bangla additions for writing currency and fractions.

Historic Characters. The characters *vocalic rr*, *vocalic l* and *vocalic ll*, both in their independent and dependent forms (U+098C, U+09C4, U+09E0..U+09E3), are only used to write Sanskrit words in the Bangla script.

Characters for Assamese. Assamese employs two letters not used for the Bangla language. The Assamese letter *ra* is represented in Unicode by U+09F0 ৰ BENGALI LETTER RA WITH MIDDLE DIAGONAL, and the Assamese letter *wa* is represented by U+09F1 ৱ BENGALI LETTER RA WITH LOWER DIAGONAL.

Assamese uses a conjunct character called *kssa*. Although *kssa* is often considered a separate letter of the alphabet, it is not separately encoded. The conjunct is represented by the sequence <U+0995 ক BENGALI LETTER KA, U+09CD ◌্ BENGALI SIGN VIRAMA, U+09B7 ষ BENGALI LETTER SSA>. This same sequence is also used to represent the Bangla letter *khinya* (or *khiya*).

Assamese uses two additional consonant-vowel ligatures formed with U+09F0 ৰ BENGALI LETTER RA WITH MIDDLE DIAGONAL, which are not used for the Bangla language. These consonant-vowel ligatures are shown in the "ligated" column in *Table 12-13*.

Table 12-13. Assamese Consonant-Vowel Combinations

	Code Points	Ligated	Non-ligated
ru	<09F0, 09C1>	ৰু	ৰু
rū	<09F0, 09C2>	ৰূ	ৰূ

Rendering Behavior. Like other Brahmic scripts in the Unicode Standard, Bangla uses the *hasant* to form conjunct characters. For example, <U+09B8 স BENGALI LETTER SA, U+09CD ্ BENGALI SIGN VIRAMA, U+0995 ক BENGALI LETTER KA> yields the conjunct স্ক SKA. For general principles regarding the rendering of the Bangla script, see the rules for rendering in *Section 12.1, Devanagari*.

Consonant-Vowel Ligatures. Some Bangla consonant plus vowel combinations have two distinct visual presentations. The first visual presentation is a traditional ligated form, in which the vowel combines with the consonant in a novel way. In the second presentation, the vowel is joined to the consonant but retains its nominal form, and the combination is not considered a ligature. These consonant-vowel combinations are illustrated in *Table 12-14*.

Table 12-14. Bangla Consonant-Vowel Combinations

	Code Points	Ligated	Non-ligated
gu	<0997, 09C1>	গু	গু
ru	<09B0, 09C1>	রু	রু
rū	<09B0, 09C2>	রূ	রূ
śu	<09B6, 09C1>	শু	শু
hu	<09B9, 09C1>	হু	হু
hṛ	<09B9, 09C3>	হৃ	হৃ

The ligature forms of these consonant-vowel combinations are traditional. They are used in handwriting and some printing. The "non-ligated" forms are more common; they are used in newspapers and are associated with modern typefaces. However, the traditional ligatures are preferred in some contexts.

No semantic distinctions are made in Bangla text on the basis of the two different presentations of these consonant-vowel combinations. However, some users consider it important that implementations support both forms and that the distinction be representable in plain text. This may be accomplished by using U+200D ZERO

WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER to influence ligature glyph selection. (See "Cursive Connection and Ligatures" in *Section 23.2, Layout Controls*.) Joiners are rarely needed in this situation. The rendered appearance will typically be the result of a font choice.

A given font implementation can choose whether to treat the ligature forms of the consonant-vowel combinations as the defaults for rendering. If the non-ligated form is the default, then ZWJ can be inserted to request a ligature, as shown in *Figure 12-12*.

Figure 12-12. Requesting Bangla Consonant-Vowel Ligature
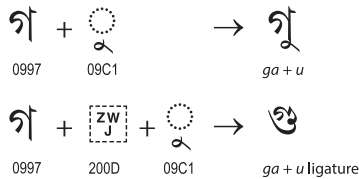
If the ligated form is the default for a given font implementation, then ZWNJ can be inserted to block a ligature, as shown in *Figure 12-13*.

Figure 12-13. Blocking Bangla Consonant-Vowel Ligature
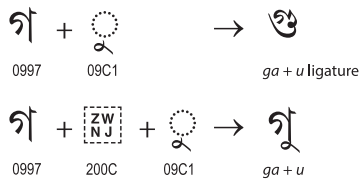
Khiya. The letter ক্ষ, known as *khiya* or *khinya*, is often considered as a distinct letter of the Bangla alphabet. However, it is not encoded separately. It is represented by the sequence <U+0995 ক BENGALI LETTER KA, U+09CD ্ BENGALI SIGN VIRAMA, U+09B7 ষ BENGALI LETTER SSA>.

Khanda Ta. In Bangla, a dead consonant *ta* makes use of a special form, U+09CE ৎ BENGALI LETTER KHANDA TA. This form is used in all contexts except where it is immediately followed by one of the consonants: *ta*, *tha*, *na*, *ba*, *ma*, *ya*, or *ra*.

Khanda ta cannot bear a vowel matra or combine with a following consonant to form a conjunct *aksara*. It can form a conjunct *aksara* only with a preceding dead consonant *ra*, with the latter being displayed with a *repha* glyph placed on the *khanda ta*.

Versions of the Unicode Standard prior to Version 4.1 recommended that *khanda ta* be represented as the sequence <U+09A4 ত BENGALI LETTER TA, U+09CD ্ BENGALI SIGN VIRAMA, U+200D ZERO WIDTH JOINER> in all circumstances.

U+09CE ৎ ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ᴋʜᴀɴᴅᴀ ᴛᴀ should instead be used explicitly in newly generated text, but users are cautioned that instances of the older representation may exist.

The Bangla syllable *tta* illustrates the usage of *khanda ta* when followed by *ta*. The syllable *tta* is normally represented with the sequence <U+09A4 *ta*, U+09CD *hasant*, U+09A4 *ta*>. That sequence will normally be displayed using a single glyph *tta* ligature, as shown in the first example in *Figure 12-14*.

Figure 12-14. Bangla Syllable *tta*
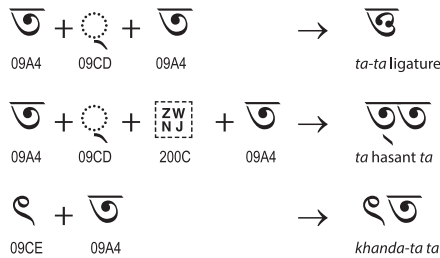
It is also possible for the sequence <*ta*, *hasant*, *ta*> to be displayed with a full *ta* glyph combined with a *hasant* glyph, followed by another full *ta* glyph ত্ত. The choice of form actually displayed depends on the display engine, based on the availability of glyphs in the font.

The Unicode Standard also provides an explicit way to show the *hasant* glyph. To do so, a ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ is inserted after the *hasant*. That sequence is always displayed with the explicit *hasant*, as shown in the second example in *Figure 12-14*.

When the syllable *tta* is written with a *khanda ta*, however, the character U+09CE ৎ ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ᴋʜᴀɴᴅᴀ ᴛᴀ is used and no *hasant* is required, as *khanda ta* is already a dead consonant. The rendering of *khanda ta* is illustrated in the third example in *Figure 12-14*.

Ya-phalaa. *Ya-phalaa* is a presentation form of U+09AF য ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ʏᴀ. Represented by the sequence <U+09CD ্ ʙᴇɴɢᴀʟɪ ꜱɪɢɴ ᴠɪʀᴀᴍᴀ, U+09AF য ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ʏᴀ>, *ya-phalaa* has a special form ্য. When combined with U+09BE া ʙᴇɴɢᴀʟɪ ᴠᴏᴡᴇʟ ꜱɪɢɴ ᴀᴀ, it is used for transcribing [æ] as in the "a" in the English word "bat." The *ya-phalaa* appears in র্যাশ [ræʃ] "rash," which provides a minimal pair with রাশ [raʃ] "a whole lot."

Ya-phalaa can be applied to initial vowels as well:

অ্যা = <0985, 09CD, 09AF, 09BE> (*a- hasant ya -aa*)

এ্যা = <098F, 09CD, 09AF, 09BE> (*e- hasant ya -aa*)

If a candrabindu or other combining mark needs to be added in the sequence, it comes at the end of the sequence. For example:

অ্যাঁ = <0985, 09CD, 09AF, 09BE, 0981> (*a- hasant ya -aa candrabindu*)

Further examples:

অ + ্ + য + া → অ্যা

এ + ্ + য + া → এ্যা

ত + ্ + য + া → ত্যা

Interaction of Repha and Ya-phalaa. The formation of the *repha* form is defined in <u>*Section 12.1, Devanagari*</u>, "Rules for Rendering," R2. Basically, the *repha* is formed when a *ra* that has the inherent vowel killed by the *hasant* begins a syllable. This scenario is shown in the following example:

র + ্ + ম → র্ম as in কর্ম (karma)

The *ya-phalaa* is a post-base form of *ya* and is formed when the *ya* is the final consonant of a syllable cluster. In this case, the previous consonant retains its base shape and the *hasant* is combined with the following *ya*. This scenario is shown in the following example:

ক + ্ + য → ক্য as in বাক্য (bakyô)

An ambiguous situation is encountered when the combination of *ra + hasant + ya* is encountered:

র + ্ + য → র্য or র্য

To resolve the ambiguity with this combination, the Unicode Standard adopts the convention of placing the character U+200D ZERO WIDTH JOINER immediately after the *ra* to obtain the *ya-phalaa*. The *repha* form is rendered when no ZWJ is present, as shown in the following example:

র + ্ + য → র্য

09B0   09CD   09AF

র + [ZWJ] + ্ + য → র্য

09B0   200D   09CD   09AF

When the first character of the cluster is not a *ra*, the *ya-phalaa* is the normal rendering of a *ya*, and a ZWJ is not necessary but can be present. Such a convention

would make it possible, for example, for input methods to consistently associate *ya-phalaa* with the sequence <ZWJ, *hasant*, *ya*>.

Jihvamuliya and Upadhmaniya. In Bangla, the voiceless velar and bilabial fricatives are represented by U+1CF5 ⌧ VEDIC SIGN JIHVAMULIYA and U+1CF6 ⌧ VEDIC SIGN UPADHMANIYA, respectively. When the signs appear with a following homorganic voiceless stop consonant, they can be rendered in a font as a stacked ligature without a virama:

$$\boxed{\text{x}} + \text{ক} \rightarrow \overset{\text{x}}{\text{ক}}$$

$$\boxed{\text{ⱳ}} + \text{প} \rightarrow \overset{\text{ⱳ}}{\text{প}}$$

The sequences can also be represented linearly by inserting a U+200C ZERO WIDTH NON-JOINER after the *jihvamuliya* or *upadhmaniya*, but before the following consonant:

$$\boxed{\text{X}} + \boxed{\substack{\text{ZW}\\\text{NJ}}} + \text{ক} \rightarrow \text{xক}$$

$$\boxed{\text{ⱳ}} + \boxed{\substack{\text{ZW}\\\text{NJ}}} + \text{প} \rightarrow \text{ⱳপ}$$

Dependent vowel signs can also be added to the stack or linear sequence. Consonant clusters containing U+1CF5 VEDIC SIGN JIHVAMULIYA and U+1CF6 VEDIC SIGN UPADHMANIYA can occur with more than two consonants, such as *ḫkra* and *ḫpra*.

Punctuation. Bangla uses punctuation marks shared across many Indic scripts, including the *danda* and *double danda* marks. In Bangla these are called the *dahri* and *double dahri*. For a description of these common punctuation marks, see *Section 12.1, Devanagari*.

Truncation. The orthography of the Bangla language makes use of U+02BC " ' " MODIFIER LETTER APOSTROPHE to indicate the truncation of words. This sign is called *urdha-comma*. Examples illustrating the use of U+02BC MODIFIER LETTER APOSTROPHE are shown in *Table 12-15*.

Table 12-15. Use of Apostrophe in Bangla

Example	Meaning
ক'রে	after, on doing (something)
'পরে ওপরে	} above

12.3 Gurmukhi

12.3.1 Gurmukhi: U+0A00–U+0A7F

The Gurmukhi script is a North Indian script used to write the Punjabi (or Panjabi) language of the Punjab state of India. Gurmukhi, which literally means "proceeding from the mouth of the Guru," is attributed to Angad, the second Sikh Guru (1504–1552 CE). It is derived from an older script called Landa and is closely related to Devanagari structurally. The script is closely associated with Sikhs and Sikhism, but it is used on an everyday basis in East Punjab. (West Punjab, now in Pakistan, uses the Arabic script.)

Encoding Principles. The Gurmukhi block is based on ISCII-1988, which makes it parallel to Devanagari. Gurmukhi, however, has a number of peculiarities described here.

The additional consonants (called *pairin bindi*; literally, "with a dot in the foot," in Punjabi) are primarily used to differentiate Urdu or Persian loan words. They include U+0A36 ਸ਼ GURMUKHI LETTER SHA and U+0A33 ਲ਼ GURMUKHI LETTER LLA, but do not include U+0A5C ੜ GURMUKHI LETTER RRA, which is genuinely Punjabi. For unification with the other scripts, ISCII-1991 considers *rra* to be equivalent to *dda+nukta*, but this decomposition is not considered in Unicode. At the same time, ISCII-1991 does not consider U+0A36 to be equivalent to <0A38, 0A3C>, or U+0A33 to be equivalent to <0A32, 0A3C>.

Two different marks can be associated with U+0902 ं DEVANAGARI SIGN ANUSVARA: U+0A02 ਂ GURMUKHI SIGN BINDI and U+0A70 ਂ GURMUKHI TIPPI. Present practice is to use *bindi* only with the dependent and independent forms of the vowels *aa*, *ii*, *ee*, *ai*, *oo*, and *au*, and with the independent vowels *u* and *uu*; *tippi* is used in the other contexts. Older texts may depart from this requirement. ISCII-1991 uses only one encoding point for both marks.

U+0A71 ੱ GURMUKHI ADDAK is a special sign to indicate that the following consonant is geminate. ISCII-1991 does not have a specific code point for addak and encodes it as a cluster. For example, the word ਪੱਗ *pagg*, "turban," can be represented with the sequence <0A2A, 0A71, 0A17> (or <pa, addak, ga>) in Unicode, while in ISCII-1991 it would be <pa, ga, virama, ga>.

U+0A75 ੵ GURMUKHI SIGN YAKASH probably originated as a subjoined form of U+0A2F ਯ GURMUKHI LETTER YA. However, because its usage is relatively rare and not entirely predictable, it is encoded as a separate character. Some modern fonts render *yakash* with the glyph ੵ, which varies from the traditional shape found in the code charts. This character should occur after the consonant to which it attaches and before any vowel sign.

U+0A51 ੑ GURMUKHI SIGN UDAAT occurs in older texts and indicates a high tone. This character should occur after the consonant to which it attaches and before any vowel sign.

Unusual Usage of Vowel Signs. In older texts, such as the *Sri Guru Granth Sahib* (the Sikh holy book), one can find typographic clusters with a vowel sign attached to a vowel letter, or with two vowel signs attached to a consonant. The most common cases are ੁ *u* attached to ੲ, as in ੳਮਾਹ and both the vowel signs ੋ and ੁ attached to a consonant, as in ਗੋੁਬਿੰਦ *goubinda*; this is used to indicate the metrical shortening of /o/ or the lengthening of /u/ depending on the context. Other combinations are attested as well, such as ਗ੍ਹਿਾਨ *ghiana*, represented by the sequence <U+0A17, U+0A4D, U+0A39, U+0A3F, U+0A3E, U+0A28>.

Because of the combining classes of the characters U+0A4B ੋ GURMUKHI VOWEL SIGN OO and U+0A41 ੁ GURMUKHI VOWEL SIGN U, the sequences <consonant, U+0A4B, U+0A41> and <consonant, U+0A41, U+0A4B> are not canonically equivalent. To avoid ambiguity in representation, the first sequence, with U+0A4B before U+0A41, should be used in such cases. More generally, when a consonant or independent vowel is modified by multiple vowel signs, the sequence of the vowel signs in the underlying representation of the text should be: left, top, bottom, right.

Unusual Positioning of bindi. Typically, when U+0A40 ੀ GURMUKHI VOWEL SIGN II and U+0A02 ੰ GURMUKHI SIGN BINDI coexist in an orthographic syllable, the *bindi* is encoded after and rendered on the right side of the *vowel sign ii*. In cases where a special left side placement of the *bindi* must be distinguished in encoding, the *bindi* can be encoded immediately preceding the *vowel sign ii* instead.

In particular, this encoding order also applies when *bindi* must appear on top of *iri* preceding *vowel sign ii*: <0A72 *iri*, 0A02 *bindi*, 0A40 *vowel sign ii*>. This sequential encoding does not conflict with the "Do Not Use" instruction about U+0A08 ਈ GURMUKHI LETTER II in *Table 12-16* because of the *bindi* inserted in between.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-16* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-16. Gurmukhi Vowel Letters

For	Use	Do Not Use
ਆ	0A06	<0A05, 0A3E>
ਇ	0A07	<0A72, 0A3F>
ਈ	0A08	<0A72, 0A40>
ਉ	0A09	<0A73, 0A41>
ਊ	0A0A	<0A73, 0A42>

ਏ	0A0F	<0A72, 0A47>
ਐ	0A10	<0A05, 0A48>
ਓ	0A13	<0A73, 0A4B>
ਔ	0A14	<0A05, 0A4C>

Tones. The Punjabi language is tonal, but the Gurmukhi script does not contain any specific signs to indicate tones. Instead, the voiced aspirates (*gha*, *jha*, *ddha*, *dha*) and the letter *ha* combine consonantal and tonal functions.

Ordering. U+0A73 ਓ GURMUKHI URA and U+0A72 ੲ GURMUKHI IRI are the first and third "letters" of the Gurmukhi syllabary, respectively. They are used as bases or bearers for some of the independent vowels, while U+0A05 ਅ GURMUKHI LETTER A is both the second "letter" and the base for the remaining independent vowels. As a result, the collation order for Gurmukhi is based on a seven-by-five grid:

- The first row is U+0A73 *ura*, U+0A05 *a*, U+0A72 *iri*, U+0A38 *sa*, U+0A39 *ha*.
- This row is followed by five main rows of consonants, grouped according to the point of articulation, as is traditional in all South and Southeast Asian scripts.
- The semiconsonants follow in the seventh row: U+0A2F *ya*, U+0A30 *ra*, U+0A32 *la*, U+0A35 *va*, U+0A5C *rra*.
- The letters with *nukta*, added later, are presented in a subsequent eighth row if needed.

Rendering Behavior. For general principles regarding the rendering of the Gurmukhi script, see the rules for rendering in *Section 12.1, Devanagari*. In many aspects, Gurmukhi is simpler than Devanagari. In modern Punjabi, there are no half-consonants, no half-forms, no *repha* (upper form of U+0930 र DEVANAGARI LETTER RA), and no real ligatures. Rules R2–R5, R11, and R14 do not apply. Conversely, the behavior for subscript RA (rules R6–R8 and R13) applies to U+0A39 ਹ GURMUKHI LETTER HA and U+0A35 ਵ GURMUKHI LETTER VA, which also have subjoined forms, called *pairin* in Punjabi. The subjoined form for RA is like a knot, while the subjoined HA and VA are written the same as the base form, without the top bar, but are reduced in size. As described in rule R13, they attach at the bottom of the base consonant, and will "push" down any attached vowel sign for U or UU. When U+0A2F ਯ GURMUKHI LETTER YA follows a dead consonant, it assumes a different form called *addha* in Punjabi, without the leftmost part, and the dead consonant returns to the nominal form, as shown in *Table 12-17*.

Table 12-17. Gurmukhi Conjuncts

ਸ	+	੍	+	ਹ	→	ਸ੍ਹ	(*mha*)	pairin ha
ਪ	+	੍	+	ਰ	→	ਪ੍ਰ	(*pra*)	pairin ra
ਦ	+	੍	+	ਵ	→	ਦ੍ਵ	(*dva*)	pairin va

ਦ	+	੍	+	ਯ	→	ਦ੍ਯ	(*dya*)	addha ya

Other letters behaved similarly in old inscriptions, as shown in *Table 12-18*.

Table 12-18. Additional Pairin and Addha Forms in Gurmukhi

ਸ	+	੍	+	ਗ	→	ਸ੍ਗ	(*sga*)	pairin ga
ਸ	+	੍	+	ਚ	→	ਸ੍ਚ	(*sca*)	pairin ca
ਸ	+	੍	+	ਟ	→	ਸ੍ਟ	(*stta*)	pairin tta
ਸ	+	੍	+	ਠ	→	ਸ੍ਠ	(*sttha*)	pairin ttha
ਸ	+	੍	+	ਤ	→	ਸ੍ਤ	(*sta*)	pairin ta
ਸ	+	੍	+	ਦ	→	ਸ੍ਦ	(*sda*)	pairin da
ਸ	+	੍	+	ਨ	→	ਸ੍ਨ	(*sna*)	pairin na
ਸ	+	੍	+	ਥ	→	ਸ੍ਥ	(*stha*)	pairin tha
ਸ	+	੍	+	ਯ	→	ਸ੍ਯ	(*sya*)	pairin ya
ਸ	+	੍	+	ਥ	→	ਸਥ	(*stha*)	addha tha
ਸ	+	੍	+	ਮ	→	ਸਮ	(*sma*)	addha ma

Older texts also exhibit another feature that is not found in modern Gurmukhi—namely, the use of a half- or reduced form for the first consonant of a cluster, whereas the modern practice is to represent the second consonant in a half- or reduced form. Joiners can be used to request this older rendering, as shown in *Table 12-19*. The reduced form of an initial U+0A30 ਰ GURMUKHI LETTER RA is similar to the Devanagari superscript RA (*repha*), but this usage is rare, even in older texts.

Table 12-19. Use of Joiners in Gurmukhi

ਸ	+	੍	+	ਵ			→	ਸ੍ਵ	(*sva*)
ਰ	+	੍	+	ਵ			→	ਰੁ	(*rva*)
ਸ	+	੍	+	ZW	+	ਵ	→	ਸਵ	(*sva*)
ਰ	+	੍	+	ZW	+	ਵ	→	ਵੈ	(*rva*)
ਸ	+	੍	+	ZW NJ	+	ਵ	→	ਸ੍ਵ	(*sva*)
ਰ	+	੍	+	ZW NJ	+	ਵ	→	ਰੁਵ	(*rva*)

A rendering engine for Gurmukhi should make accommodations for the correct positioning of the combining marks (see *Section 5.13, Rendering Nonspacing Marks*, and particularly *Figure 5-11*). This is important, for example, in the correct centering of the marks above and below U+0A28 ਨ GURMUKHI LETTER NA and U+0A20 ਠ GURMUKHI LETTER TTHA, which are laterally symmetrical. It is also important to avoid collisions between the various upper marks, vowel signs, *bindi*, and/or *addak*.

Other Symbols. The religious symbol *khanda* sometimes used in Gurmukhi texts is encoded at U+262C ☬ ADI SHAKTI in the Miscellaneous Symbols block. U+0A74 ੴ GURMUKHI EK ONKAR, which is also a religious symbol, can have different presentation forms, which do not change its meaning. The representative glyph shown the code charts is a simple form that looks like the digit one, followed by a sign based on *ura*, along with a long upper tail; other forms may be highly stylized.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with Gurmukhi are found in the Devanagari block. See *Section 12.1, Devanagari*, for more information. Punjabi also uses Latin punctuation.

12.4 Gujarati

12.4.1 Gujarati: U+0A80–U+0AFF

The Gujarati script is a North Indian script closely related to Devanagari. It is most obviously distinguished from Devanagari by not having a horizontal bar for its letter-forms, a characteristic of the older Kaithi script to which Gujarati is related. The Gujarati script is used to write the Gujarati language of the Gujarat state in India.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-20* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-20. Gujarati Vowel Letters

For	Use	Do Not Use
આ	0A86	<0A85, 0ABE>
ઍ	0A8D	<0A85, 0AC5>
એ	0A8F	<0A85, 0AC7>
ઐ	0A90	<0A85, 0AC8>
ઑ	0A91	<0A85, 0AC9>
ઓ	0A93	<0A85, 0ACB> or <0A85, 0ABE, 0AC5>
ઔ	0A94	<0A85, 0ACC> or <0A85, 0ABE, 0AC8>
◌ૉ	0AC9	<0AC5, 0ABE>

Rendering Behavior. For rendering of the Gujarati script, see the rules for rendering in *Section 12.1, Devanagari*. Like other Brahmic scripts in the Unicode Standard, Gujarati uses the virama to form conjunct characters. The virama is informally called *khoḍo*, which means "lame" in Gujarati. Many conjunct characters, as in Devanagari, lose the vertical stroke; there are also vertical conjuncts. U+0AB0 GUJARATI LETTER RA takes special forms when it combines with other consonants, as shown in *Table 12-21*.

Table 12-21. Gujarati Conjuncts

ક	+	◌	+	ષ	→	ક્ષ	(kṣa)
જ	+	◌	+	ઞ	→	જ્ઞ	(jña)
ત	+	◌	+	ય	→	ત્ય	(tya)
ટ	+	◌	+	ટ	→	ટ્ટ	(ṭṭa)
ર	+	◌	+	ક	→	ર્ક	(rka)
ક	+	◌	+	ર	→	ક્ર	(kra)

Marks for Transliteration of Arabic. The combining marks encoded in the range U+0AFA..U+0AFF are used for the transliteration of the Arabic script into Gujarati. This system of transliteration was devised in the late 19th century, and is used by Ismaili Khoja communities. These marks occur both in manuscripts and in printed materials.

The three forms of *nukta* encoded in the range U+0AFD..U+0AFF are diacritics, placed above regular Gujarati letters to create new letters corresponding to Arabic letters for non-Gujarati sounds. U+0AFF GUJARATI SIGN TWO-CIRCLE NUKTA ABOVE is used only with U+0A9D GUJARATI LETTER JHA, to transliterate the Arabic *zah*. U+0AFE GUJARATI SIGN CIRCLE NUKTA ABOVE is used with U+0A9D GUJARATI LETTER JHA to transliterate the Arabic *thal* and with U+0AB8 GUJARATI LETTER SA to transliterate the Arabic *theh*. U+0AFD GUJARATI SIGN THREE-DOT NUKTA ABOVE occurs with a number of different Gujarati letters, to transliterate a variety of Arabic letters.

U+0AFA GUJARATI SIGN SUKUN, U+0AFB GUJARATI SIGN SHADDA, and U+0AFC GUJARATI SIGN MADDAH are used to transliterate the Arabic *sukun*, *shadda*, and *maddah above*, respectively. These marks may be applied to a Gujarati letter which also uses one of the three above-base *nukta* diacritic marks. In such cases, the *nukta* occurs first in the combining sequence, followed by the *sukun*, *shadda*, or *maddah* mark. However, instead of being rendered above the *nukta* mark on the letter, the *sukun*, *shadda*, or *maddah* mark is rendered to the left of the *nukta* mark.

Punctuation. Words in Gujarati are separated by spaces. Danda and double danda marks as well as some other unified punctuation used with Gujarati are found in the Devanagari block; see *Section 12.1, Devanagari*.

12.5 Oriya (Odia)

12.5.1 Oriya: U+0B00–U+0B7F

The Oriya script is used to write the Odia language of the Odisha (Orissa) state in India, as well as minority languages such as Khondi and Santali.

Languages and scripts can be referred to in many different ways, and these terms may evolve over time. The Oriya script is an example of this: The preferred Latin transcription used in India for this script has shifted to the spelling Odia (as shown, for example, by changes to the Indian constitution). The Unicode Standard retains the traditional English spelling Oriya in discussion, to minimize the potential for confusion when referring to immutable, standardized character names in the standard, which were assigned long ago.

Special Characters. U+0B57 ORIYA AU LENGTH MARK is provided as an encoding for the right side of the surroundant vowel U+0B4C ORIYA VOWEL SIGN AU.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-22* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-22. Oriya Vowel Letters

For	Use	Do Not Use
ଆ	0B06	<0B05, 0B3E>
ଐ	0B10	<0B0F, 0B57>
ଔ	0B14	<0B13, 0B57>

Rendering Behavior. For rendering of the Oriya script, see the rules for rendering in *Section 12.1, Devanagari*. Like other Brahmic scripts in the Unicode Standard, Oriya uses the virama to suppress the inherent vowel. Oriya has a visible virama, often being a lengthening of a part of the base consonant:

 କ + ୍ → କ୍ (*k*)

The virama is also used to form conjunct consonants, as shown in *Table 12-23*.

Table 12-23. Oriya Conjuncts

କ	+	୍	+	ଷ	→	କ୍ଷ	(*kṣa*)
କ	+	୍	+	ତ	→	କ୍ତ	(*kta*)
ତ	+	୍	+	କ	→	ତ୍କ	(*tka*)

| ତ | + | ◌ୃ | + | ୟ | → | ତ୍ୟ | (*tya*) |

Consonant Forms. In the initial position in a cluster, RA is reduced and placed above the following consonant, while it is also reduced in the second position:

ର + ◌ + ପ → ର୍ପ (*rpa*)

ପ + ◌ + ର → ପ୍ର (*pra*)

Nasal and stop clusters may be written with conjuncts, or the anusvara may be used:

ଅ + ଙ + ◌ + କ → ଅଙ୍କ (*aṅka*)

ଅ + ◌ଂ + କ → ଅଂକ (*aṁka*)

Vowels. As with other scripts, some dependent vowels are rendered in front of their consonant, some appear after it, and some are placed above or below it. Some are rendered with parts both in front of and after their consonant. A few of the dependent vowels fuse with their consonants. U+0B01 ORIYA SIGN CANDRABINDU is used for nasal vowels. See *Table 12-24*.

Table 12-24. Oriya Vowel Placement

କ	+	◌ା	→	କା	(*kā*)
କ	+	◌ି	→	କି	(*ki*)
କ	+	◌ୀ	→	କୀ	(*kī*)
କ	+	◌ୁ	→	କୁ	(*ku*)
କ	+	◌ୂ	→	କୂ	(*kū*)
କ	+	◌ୃ	→	କୃ	(*kṛ*)
କ	+	ୋ◌	→	କେ	(*ke*)
କ	+	ୋ◌	→	କୈ	(*kai*)
କ	+	ୋ◌	→	କୋ	(*ko*)
କ	+	ୋ◌	→	କୌ	(*kau*)
କ	+	◌ଁ	→	କଁ	(*kaṁ*)

An orthography for the Kuvi language makes use of a macron-shaped length mark. It is displayed directly above written forms of the following three vowels to indicate their corresponding long vowels:

[o] vowel letter *a*, or inherent vowel implied by consonant letters and conjuncts

[a] vowel letter or sign *aa*

[e] vowel letter or sign *e*

This length mark is represented in text by U+0B55 ORIYA SIGN OVERLINE. It occurs in the text representation directly after the letter or sign it modifies, and after any *nukta* which is present.

Oriya VA and WA. These two letters are extensions to the basic Oriya alphabet. Because Sanskrit वन *vana* becomes Oriya ବନ *bana* in orthography and pronunciation, an extended letter U+0B35 ଵ ORIYA LETTER VA was devised by dotting U+0B2C ବ ORIYA LETTER BA for use in academic and technical text. For example, basic Oriya script cannot distinguish Sanskrit बव *bava* from बब *baba* or वव *vava*, but this distinction can be made with the modified version of *ba*. In some older sources, the glyph ଵ is sometimes found for *va*; in others, ୱ and ୱ have been shown, which in a more modern type style would be ଵ. The letter *va* is not in common use today.

In a consonant conjunct, subjoined U+0B2C ବ ORIYA LETTER BA is usually—but not always—pronounced [wa]:

> U+0B15 କ *ka* + U+0B4D ୍ *virama* + U+0B2C ବ *ba* → କ୍ବ [kwa]

> U+0B2E ମ *ma* + U+0B4D ୍ *virama* + U+0B2C ବ *ba* → ମ୍ବ [mba]

The extended Oriya letter U+0B71 ୱ ORIYA LETTER WA is sometimes used in Perso-Arabic or English loan words for [w]. It appears to have originally been devised as a ligature of ଓ *o* and ବ *ba*, but because ligatures of independent vowels and consonants are not normally used in Oriya, this letter has been encoded as a single character that does not have a decomposition. It is used initially in words or orthographic syllables to represent the foreign consonant; as a native semivowel, *virama* + *ba* is used because that is historically accurate. Glyph variants of *wa* are ୱ, ୱ, and ଓ.

Punctuation and Symbols. Danda and double danda marks as well as some other unified punctuation used with Oriya are found in the Devanagari block; see *Section 12.1, Devanagari*. The mark U+0B70 ORIYA ISSHAR is placed before names of persons who are deceased.

The sacred syllable *om* is formed by U+0B13 ORIYA LETTER O and U+0B01 ORIYA SIGN CANDRABINDU. Ligation of the two glyphs can be encouraged or discouraged by the use of U+200D ZERO WIDTH JOINER or U+200C ZERO WIDTH NON-JOINER between the two characters, as seen in *Table 12-25*. In the absence of a joiner, both the non-ligated and the ligated forms are acceptable renderings.

Table 12-25. Ligation for the Syllable *om*

ଓ	+	ZWJ	+	ଁ	→	ଓଁ or ଓଁ
ଓ	+	ZWNJ	+	ଁ	→	ଓଁ

Fraction Characters. As for many other scripts of India, Oriya has characters used to denote factional values. These were more commonly used before the advent of

decimal weights, measures, and currencies. Oriya uses six signs: three for quarter values (1/4, 1/2, 3/4) and three for sixteenth values (1/16, 1/8, and 3/16). These are used additively, with quarter values appearing before sixteenths. Thus U+0B73 ORIYA FRACTION ONE HALF followed by U+0B75 ORIYA FRACTION ONE SIXTEENTH represents the value 5/16.

12.6 Tamil

12.6.1 Tamil: U+0B80–U+0BFF

The Tamil script is descended from the South Indian branch of Brahmi. It is used to write the Tamil language of the Tamil Nadu state in India as well as minority languages such as Irula, the Dravidian language Badaga, and the Indo-European language Saurashtra. Tamil is also used in Sri Lanka, Singapore, and parts of Malaysia.

The Tamil script has fewer consonants than the other Indic scripts. When representing the "missing" consonants in transcriptions of languages such as Sanskrit or Saurashtra, superscript European digits are often used, so ⊔² = *pha*, ⊔³ = *ba*, and ⊔⁴ = *bha*. The characters U+00B2, U+00B3, and U+2074 can be used to preserve this distinction in plain text. The Grantha script is often also used by Tamil speakers to write Sanskrit because Grantha contains these missing consonants.

The Tamil script also avoids the use of conjunct consonant forms, although a few conventional conjuncts are used.

Virama (Puḷḷi). Because the Tamil encoding in the Unicode Standard is based on ISCII-1988 (Indian Script Code for Information Interchange), it makes use of the *abugida* model. An abugida treats the basic consonants as containing an inherent vowel, which can be canceled by the use of a visible mark, called a *virama* in Sanskrit. In most Brahmi-derived scripts, the placement of a virama between two consonants implies the deletion of the inherent vowel of the first consonant and causes a conjoined or subjoined consonant cluster. In those scripts, U+200C ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ is used to display a visible virama, as shown previously in the Devanagari example in *Figure 12-4*.

The situation is quite different for Tamil because the script uses very few consonant conjuncts. An orthographic cluster consisting of multiple consonants (represented by <C1, U+0BCD ◌் ᴛᴀᴍɪʟ sɪɢɴ ᴠɪʀᴀᴍᴀ, C2, …>) is normally displayed with explicit viramas, which are called *puḷḷi* in Tamil. The *puḷḷi* is typically rendered as a dot centered above the character. It occasionally appears as small circle instead of a dot, but this glyph variant should be handled by the font, and not be represented by the similar-appearing U+0B82 ◌ஂ ᴛᴀᴍɪʟ sɪɢɴ ᴀɴᴜsᴠᴀʀᴀ.

The conjuncts *kssa* and *shrii* are traditionally displayed by conjunct ligatures, as illustrated for *kssa* in *Figure 12-15*, but nowadays tend to be displayed using an explicit *puḷḷi* as well.

Figure 12-15. Kssa Ligature in Tamil

க + ் + ஷ → க்ஷ *kṣa*

To explicitly display a *puḷḷi* for such sequences, U+200C ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ can be inserted after the *puḷḷi* in the sequence of characters.

Rendering of the Tamil Script. The Tamil script is complex and requires special rules for rendering. The following discussion describes the most important features of Tamil rendering behavior. As with any script, a more complex procedure can add rendering characteristics, depending on the font and application.

> *In a font that is capable of rendering Tamil, the number of glyphs is greater than the number of Tamil characters.*

12.6.2 Tamil Vowels

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-26* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-26. Tamil Vowel Letters

For	Use	Do Not Use
ஔ	0B86	<0B85, 0BC2>

Independent Versus Dependent Vowels. In the Tamil script, the dependent vowel signs are not equivalent to a sequence of *virama + independent vowel*. For example:

ன + ி ≠ ன + ் + இ

Left-Side Vowels. The Tamil vowels U+0BC6 ெ◌, U+0BC7 ே◌, and U+0BC8 ை◌ are reordered in front of the consonant to which they are applied. When occurring in a syllable, these vowels are rendered to the left side of their consonant, as shown in *Figure 12-16*.

Figure 12-16. Tamil Vowel Reordering

க + ெ◌ → கெ

க + ே◌ → கே

க + ை◌ → கை

Two-Part Vowels. Tamil also has several vowels that consist of elements which flank the consonant to which they are applied. A sequence of two Unicode code points can be used to express equivalent spellings for these vowels, as shown in *Figure 12-17*.

Figure 12-17. Tamil Two-Part Vowels

0BCA ொ ≡ 0BC6 ெ + 0BBE ா

0BCB ோ ≡ 0BC7 ே + 0BBE ா

0BCC ௌ ≡ 0BC6 ெ + 0BD7 ௗ

In these examples, the representation on the left, which is a single code point, is the preferred form and the form in common use for Tamil.

In the process of rendering, these two-part vowels are transformed into the two separate glyphs equivalent to those on the right, which are then subject to vowel reordering, as shown in *Figure 12-18*.

Figure 12-18. Tamil Vowel Splitting and Reordering

க + ொ → கொ
க + ெ + ா → கொ
க + ோ → கோ
க + ே + ா → கோ
க + ௌ → கௌ
க + ெ + ௗ → கௌ

Even in the case where a two-part vowel occurs with a conjunct consonant or consonant cluster, the left part of the vowel is reordered around the conjunct or cluster, as shown in *Figure 12-19*.

Figure 12-19. Vowel Reordering Around a Tamil Conjunct

க + ் + ஷ + ே + ா → கேஷா *kṣō*

For either left-side vowels or two-part vowels, the ordering of the elements is unambiguous: the consonant or consonant cluster occurs first in the memory representation, followed by the vowel.

Confusable Vowels. U+0B94 TAMIL LETTER AU and U+0BCC TAMIL VOWEL SIGN AU are visually indistinguishable from two semantically unrelated sequences, as

shown in *Figure 12-20*. In the decompositions of these two vowel characters, the rightmost part is represented as the character U+0BD7 TAMIL AU LENGTH MARK, which looks exactly like the separate character, U+0BB3 TAMIL LETTER LLA.

Figure 12-20. Confusable Vowels in Tamil

0B94 ஔ ≡ 0B92 ஒ + 0BD7 ௗ ≠ 0B92 ஒ + 0BB3 ள

0BCC ௌ ≡ 0BC6 ெ + 0BD7 ௗ ≠ 0BC6 ெ + 0BB3 ள

12.6.3 Tamil Ligatures

A number of ligatures are conventionally used in Tamil. Most ligatures involve the shape taken by a consonant plus vowel sequence. A wide variety of modern Tamil words are written without a conjunct form, with a fully visible *puḷḷi*.

Ligatures with Vowel i. The vowel signs *i* ி and *ii* ீ form ligatures with the consonant *tta* ட as shown in examples 1 and 2 of *Figure 12-21*. These vowels often change shape or position slightly so as to join cursively with other consonants, as shown in examples 3 and 4 of *Figure 12-21*.

Figure 12-21. Tamil Ligatures with *i*

1 ட + ி → டி *ṭi*

2 ட + ீ → டீ *ṭī*

3 ல + ி → லி *li*

4 ல + ீ → லீ *lī*

Ligatures with Vowel u. The vowel signs *u* ு and *uu* ூ normally ligate with their consonant, as shown in *Table 12-27*. In the first column, the basic consonant is shown; the second column illustrates the ligation of that consonant with the *u* vowel sign; and the third column illustrates the ligation with the *uu* vowel sign.

Table 12-27. Tamil Ligatures with *u*

x	*x* + ு	*x* + ூ
க	கு	கூ
ங	ஙு	ஙூ
ச	சு	சூ
ஞ	ஞு	ஞூ
ட	டு	டூ
ண	ணு	ணூ

த	து	தூ
ந	நு	நூ
ன	னு	னூ
ப	பு	பூ
ம	மு	மூ
ய	யு	யூ
ர	ரு	ரூ
ற	று	றூ
ல	லு	லூ
ள	ளு	ளூ
ழ	ழு	ழூ
வ	வு	வூ

With certain consonants, ஜ, ஷ, ஸ, ஹ, and the conjunct க்ஷ, the vowel signs *u* ◌ு and *uu* ◌ூ take a distinct spacing form, as shown in *Figure 12-22*.

Figure 12-22. Spacing Forms of Tamil *u*

$$ ஜ \;+\; ◌ு \;\rightarrow\; ஜு \quad ju $$

$$ ஜ \;+\; ◌ூ \;\rightarrow\; ஜூ \quad j\bar{u} $$

Ligatures with ra. Based on typographical preferences, the consonant *ra* ர may change shape to ா, when it ligates. Such change, if it occurs, will happen only when the ா form of U+0BB0 ர TAMIL LETTER RA would not be confused with the nominal form ா of U+0BBE TAMIL VOWEL SIGN AA (namely, when ர is combined with ◌ு, ◌ி, or ◌ீ). This change in shape is illustrated in *Figure 12-23*.

Figure 12-23. Tamil Ligatures with *ra*

$$ ர \;+\; ◌ு \;\rightarrow\; ரு \quad r $$

$$ ர \;+\; ◌ி \;\rightarrow\; ரி \quad ri $$

$$ ர \;+\; ◌ீ \;\rightarrow\; ரீ \quad r\bar{i} $$

However, various governmental bodies mandate that the basic shape of the consonant *ra* ர should be used for these ligatures as well, especially in school textbooks. Media and literary publications in Malaysia and Singapore mostly use the unchanged form of *ra* ர. Sri Lanka, on the other hand, specifies the use of the changed forms shown in *Figure 12-23*.

Tamil Ligature shri. Prior to Unicode 4.1, the best mapping to represent the ligature *shri* was to the sequence <U+0BB8, U+0BCD, U+0BB0, U+0BC0>. Unicode 4.1 in 2005 added the character U+0BB6 TAMIL LETTER SHA and as a consequence, the best mapping became <U+0BB6, U+0BCD, U+0BB0, U+0BC0>. Due to slow updates to implementations, both representations are widespread in existing text. Therefore, treating both representations as equivalent sequences is recommended. *Figure 12-24* shows the two sequences.

Figure 12-24. Tamil Ligatures for *shri*
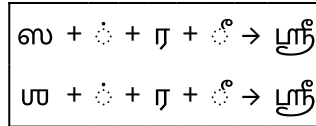
Ligatures with aa in Traditional Tamil Orthography. In traditional Tamil orthography, the vowel sign *aa* � optionally ligates with ண, ன, or ற, as illustrated in *Figure 12-25*.

Figure 12-25. Traditional Tamil Ligatures with *aa*

These ligations also affect the right-hand part of two-part vowels, as shown in *Figure 12-26*.

Figure 12-26. Traditional Tamil Ligatures with *o*

Ligatures with *ai* in Traditional Tamil Orthography. In traditional Tamil orthography, the left-side vowel sign *ai* வை is also subject to a change in form. It is rendered as ஜ when it occurs on the left side of ண, ன, ல, or ள, as illustrated in *Figure 12-27*.

Figure 12-27. Traditional Tamil Ligatures with *ai*

ண + வை → ஜண *ṇai*

ன + வை → ஜன *ṉai*

ல + வை → ஜல *lai*

ள + வை → ஜள *ḷai*

By contrast, in modern Tamil orthography, this vowel does not change its shape, as shown in *Figure 12-28*.

Figure 12-28. Vowel *ai* in Modern Tamil

ண + வை → ணை *ṇai*

Tamil aytham. The character U+0B83 TAMIL SIGN VISARGA is normally called *aytham* in Tamil. It is historically related to the *visarga* in other Indic scripts, but has become an ordinary spacing letter in Tamil. The *aytham* occurs in native Tamil words, but is frequently used as a modifying prefix before consonants used to represent foreign sounds. In particular, it is used in the spelling of words borrowed into Tamil from English or other languages.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with Tamil are found in the Devanagari block; see *Section 12.1, Devanagari*.

Numbers. Modern Tamil decimal digits are encoded at U+0BE6..U+0BEF. Note that some digits are confusable with letters, as shown in *Table 12-28*. In some Tamil fonts, the digits for two and eight look exactly like the letters *u* and *a*, respectively. In other fonts, as shown here, the shapes for the digits two and eight are adjusted to minimize confusability.

Table 12-28. Confusable Tamil Digits

U+0BE7 ௧ TAMIL DIGIT ONE	U+0B95 க TAMIL LETTER KA
U+0BE8 ௨ TAMIL DIGIT TWO	U+0B89 உ TAMIL LETTER U
U+0BED ௭ TAMIL DIGIT SEVEN	U+0B8E எ TAMIL LETTER E
U+0BEE ௮ TAMIL DIGIT EIGHT	U+0B85 அ TAMIL LETTER A

Tamil also has distinct numerals for ten, one hundred, and one thousand at U+0BF0..U+0BF2 used for historical numbers.

Use of Nukta. In addition to Tamil, several other languages of southern India are written using the Tamil script. For example, Irula is written with the Tamil script. Some of these languages contain sounds distinct from those normally written for the Tamil language. In such cases, the writing systems of these languages apply diacritic nukta marks to Tamil letters to represent their distinct sounds. For example, Irula uses a double dot nukta below represented with U+1133C GRANTHA SIGN NUKTA, and Badaga uses a single dot nukta represented by U+1133B COMBINING BINDU BELOW for some sounds.

12.6.4 Tamil Supplement: U+11FC0–U+11FFF

The Tamil Supplement block contains a set of fractions in the range U+11FC0..U+11FD4 used for generic measurement and calculations and for money. The block also includes symbols indicating various forms of measurement, old units of currency, agricultural and clerical signs, and other miscellaneous abbreviations. Most characters in this block are no longer in use, but a few appear in traditional contexts, such as on marriage invitations printed in a traditional format.

12.6.5 Tamil Named Character Sequences

Tamil is less complex than some of the other Indic scripts, and both conceptually and in processing can be treated as an atomic set of elements: consonants, stand-alone vowels, and syllables. *Table 12-29* shows these atomic elements, with the corresponding Unicode characters or sequences. In cases where the atomic elements for Tamil correspond to sequences of Unicode characters, those sequences have been added to the approved list of Unicode named character sequences. See NamedSequences.txt in the Unicode Character Database for details.

In implementations such as natural language processing, where it may be useful to treat such Tamil text elements as single code points for ease of processing, Tamil named character sequences could be mapped to code points in a contiguous segment of the Private Use Area.

In *Table 12-29*, the first row shows the transliterated representation of the Tamil vowels in abbreviated form, while the first column shows the transliterated representation of the Tamil consonants. Those row and column labels, together with identifying strings such as "TAMIL SYLLABLE" or "TAMIL CONSONANT" are concatenated to form formal names for these sequences. For example, the sequence shown in the table in the K row and the AA column, with the sequence <0B95, 0BBE>, gets the associated name TAMIL SYLLABLE KAA. The sequence shown in the table in the K row in the first column, with the sequence <0B95, 0BCD>, gets the associated name TAMIL CONSONANT K.

Details on the complete names for each element can be found in NamedSequences.txt.

Table 12-29. Tamil Vowels, Consonants, and Syllables

	A	AA	I	II	U	UU	E	EE	AI	O	OO	AU
் 0B83	அ 0B85	ஆ 0B86	இ 0B87	ஈ 0B88	உ 0B89	ஊ 0B8A	எ 0B8E	ஏ 0B8F	ஐ 0B90	ஒ 0B92	ஓ 0B93	ஔ 0B94
K க் 0B95 0BCD	க 0B95	கா 0B95 0BBE	கி 0B95 0BBF	கீ 0B95 0BC0	கு 0B95 0BC1	கூ 0B95 0BC2	கெ 0B95 0BC6	கே 0B95 0BC7	கை 0B95 0BC8	கொ 0B95 0BCA	கோ 0B95 0BCB	கௌ 0B95 0BCC
NG ங் 0B99 0BCD	ங 0B99	ஙா 0B99 0BBE	ஙி 0B99 0BBF	ஙீ 0B99 0BC0	ஙு 0B99 0BC1	ஙூ 0B99 0BC2	ஙெ 0B99 0BC6	ஙே 0B99 0BC7	ஙை 0B99 0BC8	ஙொ 0B99 0BCA	ஙோ 0B99 0BCB	ஙௌ 0B99 0BCC
C ச் 0B9A 0BCD	ச 0B9A	சா 0B9A 0BBE	சி 0B9A 0BBF	சீ 0B9A 0BC0	சு 0B9A 0BC1	சூ 0B9A 0BC2	செ 0B9A 0BC6	சே 0B9A 0BC7	சை 0B9A 0BC8	சொ 0B9A 0BCA	சோ 0B9A 0BCB	சௌ 0B9A 0BCC
NY ஞ் 0B9E 0BCD	ஞ 0B9E	ஞா 0B9E 0BBE	ஞி 0B9E 0BBF	ஞீ 0B9E 0BC0	ஞு 0B9E 0BC1	ஞூ 0B9E 0BC2	ஞெ 0B9E 0BC6	ஞே 0B9E 0BC7	ஞை 0B9E 0BC8	ஞொ 0B9E 0BCA	ஞோ 0B9E 0BCB	ஞௌ 0B9E 0BCC
TT ட் 0B9F 0BCD	ட 0B9F	டா 0B9F 0BBE	டி 0B9F 0BBF	டீ 0B9F 0BC0	டு 0B9F 0BC1	டூ 0B9F 0BC2	டெ 0B9F 0BC6	டே 0B9F 0BC7	டை 0B9F 0BC8	டொ 0B9F 0BCA	டோ 0B9F 0BCB	டௌ 0B9F 0BCC
NN ண் 0BA3 0BCD	ண 0BA3	ணா 0BA3 0BBE	ணி 0BA3 0BBF	ணீ 0BA3 0BC0	ணு 0BA3 0BC1	ணூ 0BA3 0BC2	ணெ 0BA3 0BC6	ணே 0BA3 0BC7	ணை 0BA3 0BC8	ணொ 0BA3 0BCA	ணோ 0BA3 0BCB	ணௌ 0BA3 0BCC
T த் 0BA4 0BCD	த 0BA4	தா 0BA4 0BBE	தி 0BA4 0BBF	தீ 0BA4 0BC0	து 0BA4 0BC1	தூ 0BA4 0BC2	தெ 0BA4 0BC6	தே 0BA4 0BC7	தை 0BA4 0BC8	தொ 0BA4 0BCA	தோ 0BA4 0BCB	தௌ 0BA4 0BCC
N ந் 0BA8 0BCD	ந 0BA8	நா 0BA8 0BBE	நி 0BA8 0BBF	நீ 0BA8 0BC0	நு 0BA8 0BC1	நூ 0BA8 0BC2	நெ 0BA8 0BC6	நே 0BA8 0BC7	நை 0BA8 0BC8	நொ 0BA8 0BCA	நோ 0BA8 0BCB	நௌ 0BA8 0BCC
P ப் 0BAA 0BCD	ப 0BAA	பா 0BAA 0BBE	பி 0BAA 0BBF	பீ 0BAA 0BC0	பு 0BAA 0BC1	பூ 0BAA 0BC2	பெ 0BAA 0BC6	பே 0BAA 0BC7	பை 0BAA 0BC8	பொ 0BAA 0BCA	போ 0BAA 0BCB	பௌ 0BAA 0BCC
M ம் 0BAE 0BCD	ம 0BAE	மா 0BAE 0BBE	மி 0BAE 0BBF	மீ 0BAE 0BC0	மு 0BAE 0BC1	மூ 0BAE 0BC2	மெ 0BAE 0BC6	மே 0BAE 0BC7	மை 0BAE 0BC8	மொ 0BAE 0BCA	மோ 0BAE 0BCB	மௌ 0BAE 0BCC
Y ய் 0BAF 0BCD	ய 0BAF	யா 0BAF 0BBE	யி 0BAF 0BBF	யீ 0BAF 0BC0	யு 0BAF 0BC1	யூ 0BAF 0BC2	யெ 0BAF 0BC6	யே 0BAF 0BC7	யை 0BAF 0BC8	யொ 0BAF 0BCA	யோ 0BAF 0BCB	யௌ 0BAF 0BCC
R ர் 0BB0 0BCD	ர 0BB0	ரா 0BB0 0BBE	ரி 0BB0 0BBF	ரீ 0BB0 0BC0	ரு 0BB0 0BC1	ரூ 0BB0 0BC2	ரெ 0BB0 0BC6	ரே 0BB0 0BC7	ரை 0BB0 0BC8	ரொ 0BB0 0BCA	ரோ 0BB0 0BCB	ரௌ 0BB0 0BCC
L ல் 0BB2 0BCD	ல 0BB2	லா 0BB2 0BBE	லி 0BB2 0BBF	லீ 0BB2 0BC0	லு 0BB2 0BC1	லூ 0BB2 0BC2	லெ 0BB2 0BC6	லே 0BB2 0BC7	லை 0BB2 0BC8	லொ 0BB2 0BCA	லோ 0BB2 0BCB	லௌ 0BB2 0BCC

		A	AA	I	II	U	UU	E	EE	AI	O	OO	AU
V	--	வ	வா	வி	வீ	வு	வூ	வெ	வே	வை	வொ	வோ	வௌ
	வ் 0BB5 0BCD	0BB5	0BB5 0BBE	0BB5 0BBF	0BB5 0BC0	0BB5 0BC1	0BB5 0BC2	0BB5 0BC6	0BB5 0BC7	0BB5 0BC8	0BB5 0BCA	0BB5 0BCB	0BB5 0BCC
LLL	ழ் 0BB4 0BCD	ழ 0BB4	ழா 0BB4 0BBE	ழி 0BB4 0BBF	ழீ 0BB4 0BC0	ழு 0BB4 0BC1	ழூ 0BB4 0BC2	ழெ 0BB4 0BC6	ழே 0BB4 0BC7	ழை 0BB4 0BC8	ழொ 0BB4 0BCA	ழோ 0BB4 0BCB	ழௌ 0BB4 0BCC
LL	ள் 0BB3 0BCD	ள 0BB3	ளா 0BB3 0BBE	ளி 0BB3 0BBF	ளீ 0BB3 0BC0	ளு 0BB3 0BC1	ளூ 0BB3 0BC2	ளெ 0BB3 0BC6	ளே 0BB3 0BC7	ளை 0BB3 0BC8	ளொ 0BB3 0BCA	ளோ 0BB3 0BCB	ளௌ 0BB3 0BCC
RR	ற் 0BB1 0BCD	ற 0BB1	றா 0BB1 0BBE	றி 0BB1 0BBF	றீ 0BB1 0BC0	று 0BB1 0BC1	றூ 0BB1 0BC2	றெ 0BB1 0BC6	றே 0BB1 0BC7	றை 0BB1 0BC8	றொ 0BB1 0BCA	றோ 0BB1 0BCB	றௌ 0BB1 0BCC
NNN	ன் 0BA9 0BCD	ன 0BA9	னா 0BA9 0BBE	னி 0BA9 0BBF	னீ 0BA9 0BC0	னு 0BA9 0BC1	னூ 0BA9 0BC2	னெ 0BA9 0BC6	னே 0BA9 0BC7	னை 0BA9 0BC8	னொ 0BA9 0BCA	னோ 0BA9 0BCB	னௌ 0BA9 0BCC
J	ஜ் 0B9C 0BCD	ஜ 0B9C	ஜா 0B9C 0BBE	ஜி 0B9C 0BBF	ஜீ 0B9C 0BC0	ஜு 0B9C 0BC1	ஜூ 0B9C 0BC2	ஜெ 0B9C 0BC6	ஜே 0B9C 0BC7	ஜை 0B9C 0BC8	ஜொ 0B9C 0BCA	ஜோ 0B9C 0BCB	ஜௌ 0B9C 0BCC
SH	ஶ் 0BB6 0BCD	ஶ 0BB6	ஶா 0BB6 0BBE	ஶி 0BB6 0BBF	ஶீ 0BB6 0BC0	ஶு 0BB6 0BC1	ஶூ 0BB6 0BC2	ஶெ 0BB6 0BC6	ஶே 0BB6 0BC7	ஶை 0BB6 0BC8	ஶொ 0BB6 0BCA	ஶோ 0BB6 0BCB	ஶௌ 0BB6 0BCC
SS	ஷ் 0BB7 0BCD	ஷ 0BB7	ஷா 0BB7 0BBE	ஷி 0BB7 0BBF	ஷீ 0BB7 0BC0	ஷு 0BB7 0BC1	ஷூ 0BB7 0BC2	ஷெ 0BB7 0BC6	ஷே 0BB7 0BC7	ஷை 0BB7 0BC8	ஷொ 0BB7 0BCA	ஷோ 0BB7 0BCB	ஷௌ 0BB7 0BCC
S	ஸ் 0BB8 0BCD	ஸ 0BB8	ஸா 0BB8 0BBE	ஸி 0BB8 0BBF	ஸீ 0BB8 0BC0	ஸு 0BB8 0BC1	ஸூ 0BB8 0BC2	ஸெ 0BB8 0BC6	ஸே 0BB8 0BC7	ஸை 0BB8 0BC8	ஸொ 0BB8 0BCA	ஸோ 0BB8 0BCB	ஸௌ 0BB8 0BCC
H	ஹ் 0BB9 0BCD	ஹ 0BB9	ஹா 0BB9 0BBE	ஹி 0BB9 0BBF	ஹீ 0BB9 0BC0	ஹு 0BB9 0BC1	ஹூ 0BB9 0BC2	ஹெ 0BB9 0BC6	ஹே 0BB9 0BC7	ஹை 0BB9 0BC8	ஹொ 0BB9 0BCA	ஹோ 0BB9 0BCB	ஹௌ 0BB9 0BCC
KSS	க்ஷ் 0B95 0BCD 0BB7 0BCD	க்ஷ 0B95 0BCD 0BB7	க்ஷா 0B95 0BCD 0BB7 0BBE	க்ஷி 0B95 0BCD 0BB7 0BBF	க்ஷீ 0B95 0BCD 0BB7 0BC0	க்ஷு 0B95 0BCD 0BB7 0BC1	க்ஷூ 0B95 0BCD 0BB7 0BC2	க்ஷெ 0B95 0BCD 0BB7 0BC6	க்ஷே 0B95 0BCD 0BB7 0BC7	க்ஷை 0B95 0BCD 0BB7 0BC8	க்ஷொ 0B95 0BCD 0BB7 0BCA	க்ஷோ 0B95 0BCD 0BB7 0BCB	க்ஷௌ 0B95 0BCD 0BB7 0BCC

SHRII	ஶ்ரீ 0BB6 0BCD 0BB0 0BC0

12.7 Telugu

12.7.1 Telugu: U+0C00–U+0C7F

The Telugu script is a South Indian script used to write the Telugu language of the Andhra Pradesh state in India as well as minority languages such as Gondi (Adilabad and Koi dialects) and Lambadi. The script is also used in Maharashtra, Odisha (Orissa), Madhya Pradesh, and West Bengal. The Telugu script became distinct by the thirteenth century CE and shares ancestors with the Kannada script.

Vowels. Telugu vowel letters and vowel signs are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-30* shows the letters and signs that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-30. Telugu Vowels

For	Use	Do Not Use
ఓ	0C13	<0C12, 0C55>
ఔ	0C14	<0C12, 0C4C>
ీ	0C40	<0C3F, 0C55>
ే	0C47	<0C46, 0C55>
ో	0C4B	<0C4A, 0C55>

Rendering Behavior. Telugu script rendering is similar to that of some other Brahmic scripts in the Unicode Standard—in particular, the Kannada script. (See *Section 12.8, Kannada*.) Many Telugu letters have a v-shaped headstroke, which is a structural mark corresponding to the horizontal bar in Devanagari and the arch in Oriya. When a virama (called *virāmamu* in Telugu) or certain vowel signs are added to a letter with this headstroke, it is replaced:

U+0C15 క *ka* + U+0C4D ్ *virama* → క్ (*k*)

U+0C15 క *ka* + U+0C3F ి *vowel sign i* → కి (*ki*)

Telugu consonant clusters are most commonly represented by a subscripted, and often transformed, consonant glyph for the second element of the cluster:

U+0C17 గ *ga* + U+0C4D ్ *virama* + U+0C17 గ *ga* → గ్గ (*gga*)

U+0C15 క *ka* + U+0C4D ్ *virama* + U+0C15 క *ka* → క్క (*kka*)

U+0C15 క *ka* + U+0C4D ్ *virama* + U+0C2F య *ya* → క్య (*kya*)

U+0C15 ಕ *ka* + U+0C4D ◌్ *virama* + U+0C37 ಷ *ssa* → క్ష (*kṣa*)

U+200C ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ is used to prevent U+0C4D ᴛᴇʟᴜɢᴜ sɪɢɴ ᴠɪʀᴀᴍᴀ from subscripting a following letter:

U+0C15 ಕ *ka* + U+0C4D ◌్ *virama* + U+200C 🆉🆆🅽🅹 ZWNJ + U+0C15 ಕ *ka* → క్క (*k.ka*)

Nakāra-Pollu. A distinct form ೯ of a vowelless U+0C28 న ᴛᴇʟᴜɢᴜ ʟᴇᴛᴛᴇʀ ɴᴀ appears in older Telugu texts, and is known as *nakāra-pollu*. This form is represented by a separate character, U+0C5D ೯ ᴛᴇʟᴜɢᴜ ʟᴇᴛᴛᴇʀ ɴᴀᴋᴀᴀʀᴀ ᴘᴏʟʟᴜ. The related form regularly used in modern texts takes an ordinary virama-joined shape న్, as other consonants do, and thus is represented by the sequence <U+0C28 న *na*, U+0C4D ◌్ *virama*>.

Prior to Unicode 14.0, these two distinct forms were treated as glyphic variants of that regular sequence <U+0C28 న *na*, U+0C4D ◌్ *virama*>, handled at the font level.

Reph. In modern Telugu, U+0C30 ᴛᴇʟᴜɢᴜ ʟᴇᴛᴛᴇʀ ʀᴀ behaves in the same manner as most other initial consonants in a consonant cluster. That is, the *ra* appears in its nominal form, and the second consonant takes the C2-conjoining or subscripted form:

U+0C30 ర *ra* + U+0C4D ◌్ *virama* + U+0C2E మ *ma* → ర్మ (*rma*)

However, in older texts, U+0C30 ᴛᴇʟᴜɢᴜ ʟᴇᴛᴛᴇʀ ʀᴀ takes the reduced (or *reph*) form ೯ when it appears first in a consonant cluster, and the following consonant maintains its nominal form:

U+0C30 ర *ra* + U+0C4D ◌్ *virama* + U+0C2E మ ma → మ౯ (*rma*)

U+200D ᴢᴇʀᴏ ᴡɪᴅᴛʜ ᴊᴏɪɴᴇʀ is placed immediately after the *virama* to render the *reph* explicitly in modern texts:

U+0C30 ర *ra* + U+0C4D ◌్ *virama* + U+200D 🆉🆆🅹 ZWJ + U+0C2E మ *ma* → మ౯

To prevent display of a *reph*, U+200D ᴢᴇʀᴏ ᴡɪᴅᴛʜ ᴊᴏɪɴᴇʀ is placed after the *ra*, but preceding the *virama*:

U+0C30 ర *ra* + U+200D 🆉🆆🅹 ZWJ + U+0C4D ◌్ *virama* + U+0C2E మ *ma* → ర్మ

Special Characters. U+0C55 ᴛᴇʟᴜɢᴜ ʟᴇɴɢᴛʜ ᴍᴀʀᴋ is provided as an encoding for the distinguishing element appearing in certain letters and signs, however, this character is not used in ordinary representation of Telugu texts. See "Vowel Letters" earlier in this section for more information. U+0C56 ᴛᴇʟᴜɢᴜ ᴀɪ ʟᴇɴɢᴛʜ ᴍᴀʀᴋ is provided as an encoding for the second element of the surroundrant vowel U+0C48

TELUGU VOWEL SIGN AI. The length marks are both nonspacing characters. For a detailed discussion of the use of two-part vowels, see "Two-Part Vowels" in *Section 12.6, Tamil*.

For scholarly orthographies in which a horizontal line below is used to denote an alternative vowel or consonant for a syllable, U+0952 DEVANAGARI STRESS SIGN ANUDATTA is recommended to represent the line analogously to a *svara* in an orthographic syllable. For the encoding order of *svaras*, see R10 of "Rendering Devanagari" in *Section 12.1, Devanagari*.

Nukta. U+0C3C TELUGU SIGN NUKTA is a mark placed under letters to indicate additional sounds from Tamil and Perso-Arabic languages. It may display as a large dot or as a ring, and is typically placed low enough to avoid confusion and collision with the differentiating "teardrop" that occurs under many Telugu letters. The representative glyph in the code chart is shown with the ring form to minimize accidental confusability in implementations.

Fractions. Prior to the adoption of the metric system, Telugu fractions were used as part of the system of measurement. Telugu fractions are quaternary (base-4), and use eight marks, which are conceptually divided into two sets. The first set represents odd-numbered negative powers of four in fractions. The second set represents even-numbered negative powers of four in fractions. Different zeros are used with each set. The zero from the first set is known as *haḷḷi*, U+0C78 TELUGU FRACTION DIGIT ZERO FOR ODD POWERS OF FOUR. The zero for the second set is U+0C66 TELUGU DIGIT ZERO.

Punctuation. Danda and double danda are used primarily in the domain of religious texts to indicate the equivalent of a comma and full stop, respectively. The danda and double danda marks as well as some other unified punctuation used with Telugu are found in the Devanagari block; see *Section 12.1, Devanagari*.

12.8 Kannada

12.8.1 Kannada: U+0C80–U+0CFF

The Kannada script is a South Indian script. It is used to write the Kannada (or Kanarese) language of the Karnataka state in India and to write minority languages such as Tulu. The Kannada language is also used in many parts of Tamil Nadu, Kerala, Andhra Pradesh, and Maharashtra. This script is very closely related to the Telugu script both in the shapes of the letters and in the behavior of conjunct consonants. The Kannada script also shares many features common to other Indic scripts. See *Section 12.1, Devanagari*, for further information.

The Unicode Standard follows the ISCII layout for encoding, which also reflects the traditional Kannada alphabetic order.

12.8.2 Principles of the Kannada Script

Like Devanagari and related scripts, the Kannada script employs a halant, which is also known as a virama or vowel omission sign, U+0CCD ☞ KANNADA SIGN VIRAMA. The halant nominally serves to suppress the inherent vowel of the consonant to which it is applied. The halant functions as a combining character. When a consonant loses its inherent vowel by the application of halant, it is known as a dead consonant. The dead consonants are the presentation forms used to depict the consonants without an inherent vowel. Their rendered forms in Kannada resemble the full consonant with the horn replaced by the halant sign. In contrast, a live consonant is a consonant that retains its inherent vowel or is written with an explicit dependent vowel sign. The dead consonant is defined as a sequence consisting of a consonant letter followed by a halant. The default rendering for a dead consonant is to position the halant as a combining mark bound to the consonant letterform.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-31* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-31. Kannada Vowel Letters

For	Use	Do Not Use
ಊ	0C8A	<0C89, 0CBE>
ಔ	0C94	<0C92, 0CCC>
ೠ	0CE0	<0C8B, 0CBE>

Consonant Conjuncts. Kannada is also noted for a large number of consonant conjunct forms that serve as ligatures of two or more adjacent forms. This use of liga-

tures takes place in the context of a consonant cluster. A written consonant cluster is defined as a sequence of characters that represent one or more dead consonants followed by a normal live consonant. A separate and unique glyph corresponds to each part of a Kannada consonant conjunct. Most of these glyphs resemble their original consonant forms—many without the implicit vowel sign, wherever applicable.

In Kannada, conjunct formation tends to be graphically regular, using the following pattern:

- The first consonant of the cluster is rendered with the implicit vowel or a different dependent vowel appearing as the terminal element of the cluster.

- The remaining consonants (consonants between the first consonant and the terminal vowel element) appear in conjunct consonant glyph forms in phonetic order. They are generally depicted directly below or to the lower right of the first consonant.

A Kannada script font contains the conjunct glyph components, but they are not encoded as separate Unicode characters because they are simply ligatures. Kannada script rendering software must be able to map appropriate combinations of characters in context to the appropriate conjunct glyphs in fonts.

> *In a font that is capable of rendering Kannada, the number of glyphs is greater than the number of encoded Kannada characters.*

Special Characters. U+0CD5 ಕ KANNADA LENGTH MARK is provided as an encoding for the right side of the two-part vowel U+0CC7 ಕೆ KANNADA VOWEL SIGN EE should it be necessary for processing. Likewise, U+0CD6 ಕ KANNADA AI LENGTH MARK is provided as an encoding for the right side of the two-part vowel U+0CC8 ಕೈ KANNADA VOWEL SIGN AI. The Kannada two-part vowels actually consist of a nonspacing element above the consonant letter and one or more spacing elements to the right of the consonant letter. These two length marks have no independent existence in the Kannada writing system and do not play any part as independent codes in the traditional collation order.

Kannada Letter LLLA. U+0CDE ೞ KANNADA LETTER FA is actually an archaic Kannada letter that is transliterated in Dravidian scholarship as *ẓ*, *ḷ*, or *ṛ*. This form should have been named "LLLA", rather than "FA", so the name in this standard is simply a mistake. A formal name alias KANNADA LETTER LLLA has been added to the Unicode Character Database for this character, to clarify its identity. Collations should treat U+0CDE as following U+0CB3 KANNADA LETTER LLA.

The *letter llla* has not been actively used in writing the Kannada language since the end of the tenth century. However, the letter does have modern use in writing the closely related Badaga language. Badaga is noteworthy for having distinct

retroflexion in its vowel system, and a subjoined form of U+0CDE is often seen in Badaga written documents, to indicate retroflexed pronunciation of the vowel in a syllable. This subjoined form of U+0CDE may occur below consonants, but it also may be subjoined to an independent vowel, to indicate retroflexion of that vowel. In either case, the subjoined form of U+0CDE should be represented by a sequence including U+0CCD ᴋᴀɴɴᴀᴅᴀ ꜱɪɢɴ ᴠɪʀᴀᴍᴀ. Implementations of the Kannada script need to be aware that these sequences involving independent vowels followed by virama and U+0CDE are valid and required in orthographies for Badaga. Examples of the use of subjoined U+0CDE to indicate retroflexion, both for independent vowel letters and for dependent vowels, are shown in <u>Figure 12-29</u>.

Figure 12-29. Indicating Retroflexion in Badaga Vowels
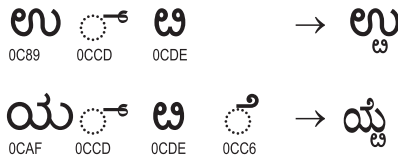
12.8.3 Rendering Kannada

Plain text in Kannada is generally stored in phonetic order; that is, a CV syllable with a dependent vowel is always encoded as a consonant letter C followed by a vowel sign V in the memory representation. This order is employed by the ISCII standard and corresponds to the phonetic and keying order of textual data.

Explicit Virama (Halant). Normally, a halant character creates dead consonants, which in turn combine with subsequent consonants to form conjuncts. This behavior usually results in a halant sign not being depicted visually. Occasionally, this default behavior is not desired when a dead consonant should be excluded from conjunct formation, in which case the halant sign is visibly rendered. To accomplish this, U+200C ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ is introduced immediately after the encoded dead consonant that is to be excluded from conjunct formation. See <u>Section 12.1, Devanagari</u>, for examples.

Vowelless NA. A special form, ℇ, of a vowelless *na* appears in older Kannada texts, distinct from the usual form of the vowelless *na* in modern texts: ನ್. The historic form is represented by a separate character, U+0CDD ᴋᴀɴɴᴀᴅᴀ ʟᴇᴛᴛᴇʀ ɴᴀᴋᴀᴀʀᴀ ᴘᴏʟʟᴜ. This character is named after the analogous Telugu form, *nakāra-pollu*, because there is no conventional term for this form in Kannada. Prior to Unicode 14.0, these two forms were treated as glyphic variants of <U+0CA8 ᴋᴀɴɴᴀᴅᴀ ʟᴇᴛᴛᴇʀ ɴᴀ, U+0CCD ᴋᴀɴɴᴀᴅᴀ ꜱɪɢɴ ᴠɪʀᴀᴍᴀ>, handled at the font level.

Consonant Clusters Involving RA. Whenever a consonant cluster is formed with the U+0CB0 ರ ᴋᴀɴɴᴀᴅᴀ ʟᴇᴛᴛᴇʀ ʀᴀ as the first component of the consonant cluster,

the letter *ra* is depicted with two different presentation forms: one as the initial element and the other as the final display element of the consonant cluster.

U+0CB0 ರ *ra* + U+0CCD ್ *halant* + U+0C95 ಕ *ka* → ರ್ಕ *rka*

U+0CB0 ರ *ra* + 🔲 + U+0CCD ್ *halant* + U+0C95 ಕ *ka* → ರ್ಕ *rka*

U+0C95 ಕ *ka* + U+0CCD ್ *halant* + U+0CB0 ರ *ra* → ಕ್ರ *kra*

Jihvamuliya and Upadhmaniya. Voiceless velar and bilabial fricatives in Kannada are represented by U+0CF1 KANNADA SIGN JIHVAMULIYA and U+0CF2 KANNADA SIGN UPADHMANIYA, respectively. When the signs appear with a following homorganic voiceless stop consonant, the combination should be rendered in the font as a stacked ligature, without a virama:

U+0CF1 ☒ *jihvamuliya* + U+0C95 ಕ *ka* → ☒

U+0CF2 ☒ *upadhmaniya* + U+0CAB ಫ *pha* → ಫ

Dependent vowels signs can also be added to the stack:

U+0CF1 ☒ *jihvamuliya* + U+0C95 ಕ *ka* + U+0CBF ಿ *vowel sign i* → ☒

Modifier Mark Rules. In addition to the vowel signs, one or more types of combining marks may be applied to a component of a written syllable or the syllable as a whole. If the consonant represents a dead consonant, then the nukta should precede the halant in the memory representation. The nukta is represented by a double-dot mark, U+0CBC ಼ KANNADA SIGN NUKTA. Two such modified consonants are used in the Kannada language: one representing the syllable *za* and one representing the syllable *fa*.

Avagraha Sign. A spacing mark, U+0CBD ಽ KANNADA SIGN AVAGRAHA, is used when rendering Sanskrit texts.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with this script are found in the Devanagari block; see *Section 12.1, Devanagari*.

12.9 Malayalam

12.9.1 Malayalam: U+0D00–U+0D7F

The Malayalam script is a South Indian script used to write the Malayalam language of the Kerala state. Malayalam is a Dravidian language like Kannada, Tamil, and Telugu. Throughout its history, it has absorbed words from Tamil, Sanskrit, Arabic, and English.

The shapes of Malayalam letters closely resemble those of Tamil. Malayalam, however, has a very full and complex set of conjunct consonant forms.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 12-32* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 12-32. Malayalam Vowel Letters

For	Use	Do Not Use
ഈ	0D08	<0D07, 0D57>
ഊ	0D0A	<0D09, 0D57>
ഐ	0D10	<0D0E, 0D46>
ഓ	0D13	<0D12, 0D3E>
ഔ	0D14	<0D12, 0D57>

Two-Part Vowels. The Malayalam script uses several two-part vowel characters. In modern times, the dominant practice is to write the dependent form of the *au* vowel using only "ൗ", which is placed on the right side of the consonant it modifies; such texts are represented in Unicode using U+0D57 MALAYALAM AU LENGTH MARK. In the past, this dependent form was written using both "െ" on the left side and "ൗ" on the right side; U+0D4C MALAYALAM VOWEL SIGN AU can be used for documents following this earlier tradition. This historical simplification started much earlier than the orthographic reforms mentioned in the text that follows.

For a detailed discussion of the use of two-part vowels, see "Two-Part Vowels" in *Section 12.6, Tamil*.

Historic and Scholarly Characters. U+0D5F MALAYALAM LETTER ARCHAIC II represents an earlier form for the vowel letter *ii*. Characters for the letters and signs of *vocalic rr,* *vocalic l*, and *vocalic ll*, as well as *candrabindu* and *avagraha*, are only used in Sanskrit texts. U+0D54..U+0D56 are rarely used *chillu* forms, found only in historical materials.

U+0D3B MALAYALAM SIGN VERTICAL BAR VIRAMA and U+0D3C MALAYALAM SIGN CIRCULAR VIRAMA are two specific forms of viramas found in historical materials. They were used to indicate a pure consonant in different orthographies. U+0D00 MALAYALAM SIGN COMBINING ANUSVARA ABOVE is used in certain Prakrit texts, where the ordinary *anusvara* indicates gemination of the following consonant.

U+0D3A MALAYALAM LETTER TTTA and U+0D29 MALAYALAM LETTER NNNA are used in scholarly orthographies for transcribing the Malayalam language in a phonetically accurate way. They represent the alveolar plosive and nasal, respectively. The letter *nnna* is parallel to U+0BA9 TAMIL LETTER NNNA.

Suriyani Malayalam. The Suriyani dialect of Malayalam is written using the Syriac script. It is also called Garshuni (Karshoni) or Syriac Malayalam. This usage requires eleven additional letters encoded in the Syriac Supplement block (U+0860..U+086F) to represent the sounds of Malayalam. The dialect was widely used by the St. Thomas Christians living in Kerala, India, in the 19th century.

12.9.2 Malayalam Orthographic Reform

In the 1970s and 1980s, Malayalam underwent orthographic reform due to printing difficulties. The treatment of the combining vowel signs *u* and *uu* was simplified at this time. These vowel signs had previously been represented using special cluster graphemes where the vowel signs were fused beneath their consonants, but in the reformed orthography they are represented by spacing characters following their consonants. *Table 12-33* lists a variety of consonants plus the *u* or *uu* vowel sign, yielding a syllable. Each syllable is shown as it would be displayed in the older orthography, contrasted with its display in the reformed orthography.

Table 12-33. Malayalam Orthographic Reform

Syllable		Older Orthography	Reformed Orthography
ku	ക + ◌ു	കു	കു
gu	ഗ + ◌ു	ഗു	ഗു
chu	ഛ + ◌ു	ഛു	ഛു
ju	ജ + ◌ു	ജു	ജു
ṇu	ണ + ◌ു	ണു	ണു
tu	ത + ◌ു	തു	തു
nu	ന + ◌ു	നു	നു
bhu	ഭ + ◌ു	ഭു	ഭു
ru	ര + ◌ു	രു	രു
śu	ശ + ◌ു	ശു	ശു
hu	ഹ + ◌ു	ഹു	ഹു

kū	ക + ◌ൂ	ക‌ൂ	കൂ
gū	ഗ + ◌ൂ	ഗൂ	ഗൂ
chū	ഛ + ◌ൂ	ഛൂ	ഛൂ
jū	ജ + ◌ൂ	ജൂ	ജൂ
ṇū	ണ + ◌ൂ	ണൂ	ണൂ
tū	ത + ◌ൂ	തൂ	തൂ
nū	ന + ◌ൂ	നൂ	നൂ
bhū	ഭ + ◌ൂ	ഭൂ	ഭൂ
rū	ര + ◌ൂ	രൂ	രൂ
śū	ശ + ◌ൂ	ശൂ	ശൂ
hū	ഹ + ◌ൂ	ഹൂ	ഹൂ

12.9.3 Rendering Malayalam

Candrakkala. As is the case for many other Brahmi-derived scripts in the Unicode Standard, Malayalam uses a virama character to form consonant conjuncts. The virama sign itself is known as *candrakkala* in Malayalam. *Table 12-34* provides a variety of examples of consonant conjuncts. There are both horizontal and vertical conjuncts, some of which ligate, and some of which are merely juxtaposed.

Table 12-34. Malayalam Conjuncts

ക	+	◌്	+	ഷ	→	ക്ഷ	*(kṣa)*
ക	+	◌്	+	ക	→	ക്ക	*(kka)*
ജ	+	◌്	+	ഞ	→	ജ്ഞ	*(jña)*
ട	+	◌്	+	ട	→	ട്ട	*(ṭṭa)*
പ	+	◌്	+	പ	→	പ്പ	*(ppa)*
ച	+	◌്	+	ഛ	→	ച്ഛ	*(ccha)*
ബ	+	◌്	+	ബ	→	ബ്ബ	*(bba)*
ന	+	◌്	+	യ	→	ന്യ	*(nya)*
പ	+	◌്	+	ര	→	പ്ര	*(pra)*
ശ	+	◌്	+	വ	→	ശ്വ	*(śva)*

When the *candrakkala* sign is visibly shown in Malayalam, it indicates either the suppression of the preceding vowel or its replacement with a neutral vowel sound. This sound is often called "half-u" or *samvruthokaram*. In various orthographies this sound is typically spelled with either a vowel sign -u followed by *candrakkala* or a *candrakkala* alone. In vernacular orthographies, *candrakkala* can also be seen on an independent vowel letter or preceding an *anusvara*. In all cases, the *candrakkala* sign is represented by the character U+0D4D MALAYALAM SIGN VIRAMA, which follows any vowel sign that may be present and precedes any *anusvara* that may be present.

Implementations need to pay careful attention to correctly shape a Malayalam ortho-graphic syllable when U+0D4D occurs in such locations. Examples are shown in *Table 12-35*.

<div align="center">

Table 12-35. Candrakkala Examples

</div>

പാലു്	/pālə/ milk	0D2A, 0D3E, 0D32, 0D41, 0D4D
എ്ന്നാ	/ənnā/ on which day? (vernacular)	0D0E, 0D4D, 0D28, 0D4D, 0D28, 0D3E
ഐശീല്ം	/aiśīləm/ than ice (vernacular)	0D10, 0D36, 0D40, 0D32, 0D4D, 0D02

Explicit Candrakkala. The sequence <C1, virama, ZWNJ, C2>, where C1 and C2 are consonants, may be used to request display with an explicit visible *candrakkala*, instead of the default conjunct form. See *Table 12-36* for an example. This convention is consistent with the use of this sequence in other Indic scripts.

Requesting Traditional Ligatures. The sequence <C1, ZWJ, virama, C2> may be used to request traditional ligatures, even if the current font defaults to the conjuncts appropriate for the reformed orthography. When such sequences occur, a closed or cursively connected ligature should be displayed, if available. See *Table 12-36* for examples. This convention is consistent with the use of this sequence in some other Indic scripts, such as Kannada, Oriya, and Telugu.

Requesting Open Forms of Conjuncts. The sequence <C1, ZWNJ, virama, C2> may be used to request open ligatures or those used in the reformed orthography, even if the current font defaults to the conjuncts appropriate for the traditional orthography. When such sequences occur, an open or disconnected conjunct form should be displayed, if available. See *Table 12-36* for examples. Note that such sequences are defined for Malayalam only, and are left undefined for other Indic scripts.

<div align="center">

Table 12-36. Use of Joiners in Malayalam

</div>

ക	+	്			+	ര	→	ക്ര or ക്ര	*(kra)*
സ	+	്			+	ക	→	സ്ക or സ്ക	*(ska)*
ത	+	്			+	സ	→	ത്സ or ത്സ	*(tsa)*
ഴ	+	്			+	വ	→	ഴ്വ or ഴ്വ or ഴ്വ	*(lva)*
യ	+	്			+	യ	→	യ്യ	*(yya)*
ക	+	്	+	ZWNJ	+	ര	→	ക്‌ര	*(kra)*
ക	+	ZWJ	+	്	+	ര	→	ക്ര	*(kra)*
സ	+	ZWJ	+	്	+	ക	→	സ്ക	*(ska)*
ത	+	ZWJ	+	്	+	സ	→	ത്സ	*(tsa)*
ഴ	+	ZWJ	+	്	+	വ	→	ഴ്വ	*(lva)*
ക	+	ZWNJ	+	്	+	ര	→	ക്ര	*(kra)*
ഴ	+	ZWNJ	+	്	+	വ	→	ഴ്വ	*(lva)*

ഗ	+	ZWNJ	+	◌്	+	ഗ	→	ഗ്യ		*(yya)*

Anusvara. The *anusvara* can be seen multiple times after vowels, whether independent letters or dependent vowel signs, as in ഈംഠഠഠ <0D08, 0D02, 0D02, 0D02, 0D02>. Vowel signs can also be seen after digits, as in 355ം <0033, 0035, 0035, 0D3E, 0D02>. More generally, rendering engines should be prepared to handle Malayalam letters (including vowel letters), digits (both European and Malayalam), U+002D HYPHEN-MINUS, U+00A0 NO-BREAK SPACE and U+25CC DOTTED CIRCLE as base characters for the Malayalam vowel signs, U+0D4D MALAYALAM SIGN VIRAMA, U+0D02 MALAYALAM SIGN ANUSVARA, and U+0D03 MALAYALAM SIGN VISARGA. They should also be prepared to handle multiple combining marks on those bases.

Dot Reph. U+0D4E MALAYALAM LETTER DOT REPH is used to represent the dead consonant form of U+0D30 MALAYALAM LETTER RA, when it is displayed as a dot or small vertical stroke above the consonant that follows it in logical order. It has the character properties of a letter rather than those of a combining mark, but special behavior is required in implementations. Conceptually, *dot reph* is analogous to the sequence <*ra, virama*> which, in many Indic scripts, is rendered as a *reph* mark over the following consonant. This same behavior is expected for *dot reph*: it should be rendered as a mark over the following consonant. In standard Malayalam, the sequence <*ra, virama*> would normally occur only within the sequence <*ra, virama, ya*>, which should be rendered as the nominal form of *ra* with a conjoining form of *ya*.

The sequence <*ra, virama, ZWJ*> is not used to represent the *dot reph*, because that sequence has considerable preexisting usage to represent the *chillu* form of *ra*, prior to the encoding of the *chillu* form as a distinct character, U+0D7C MALAYALAM LETTER CHILLU RR.

The Malayalam *dot reph* was in common print usage until 1970, but has fallen into disuse. Words that formerly used *dot reph* on a consonant are now spelled instead with a *chillu-rr* form preceding the consonant. (See the following discussion of *chillu* characters.) The *dot reph* form is predominantly used by those who completed elementary education in Malayalam prior to 1970.

Chillu Forms. The nine characters, U+0D54..U+0D56 and U+0D7A..U+0D7F, encode dead consonants (those without an inherent vowel) known as *chillu* or *cillakṣaram*. In Malayalam language text, *chillu* forms never start a word. C*hillu-nn*, -*n*, -*rr*, -*l*, and -*ll* are quite common; *chillu-k* is relatively rare in contemporary usage; *chillu-m*, -*y*, and -*lll* are found only in historical texts.

For backward-compatibility issues regarding the representation of *chillu* forms, see the discussion of legacy *chillu* sequences later in this section.

Although *chillus* are typically written alone, they may graphically behave like ordinary consonant letters. See <u>Table 12-37</u> for examples of conjuncts involving *chillus*. The *chillu*-involving conjuncts are encoded graphically: the graphic component bearing the ligated *chillu* tail is analyzed as a *chillu* character, and then stacking or ligating between characters is requested by U+0D4D MALAYALAM SIGN VIRAMA. Dependent signs such as vowel signs and *candrakkala* can be applied to both stand-alone *chillus* and *chillu*-involving conjuncts, just as they are applied to ordinary consonant letters and conjuncts.

Among the examples shown in <u>Table 12-37</u>, only the second conjunct, �൫ /ṉta/, is used in modern Malayalam text. See "Special Cases Involving *rra*" later in this section for how to deal with the contrast between this conjunct and a phonetically related side-by-side form, ൻറ.

<div align="center">

Table 12-37. Malayalam Conjuncts Involving *Chillus*

൩	0D7A *chillu nn*, 0D4D *virama*, 0D28 *na*	/ṇna/
൫	0D7B *chillu n*, 0D4D *virama*, 0D31 *rra*	/ṉta/
ൻ	0D28 *na*, 0D4D *virama*, 0D7B *chillu n*	/nn/
൰	0D7D *chillu l*, 0D4D *virama*, 0D2A *pa*	/lpa/
൱	0D7E *chillu ll*, 0D4D *virama*, 0D35 *va*	/ḷva/

</div>

U+0D3B MALAYALAM SIGN VERTICAL BAR VIRAMA is not used to form *chillus*. It only represents a vowel-killing vertical stroke that is identifiable as a separate stroke, either striking through or placed above the modified letter.

Special Cases Involving rra. There are a number of textual representation and reading issues involving the letter *rra*. These issues are discussed here and tables of explicit examples are presented.

The letter റ *rra* is normally read /ṟa/. Repetition of that sound is naturally written by repeating the letter: ററ. Each occurrence can bear a vowel sign.

The same repetition of the letter *rra* as ററ is also used for /ṭṭa/, which can be unambiguously represented by ??. The sequence of two റ letters fundamentally behaves as a digraph in this instance. The digraph can bear a vowel sign in which case the digraph as a whole acts graphically as an atom: a left vowel part goes to the left of the digraph and a right vowel part goes to the right of the digraph. Historically, the side-by-side form was used until around 1960 when the stacked form began appearing and supplanted the side-by-side form.

As a consequence the graphical sequence ററ in text is ambiguous in reading. The reader must generally use the context to understand if ററ is read /ṟaṟa/ or /ṭṭa/. It is only when a vowel part appears between the two റ that the reading cannot be /ṭṭa/. Note that similar situations are common in many other orthographies. For example,

th in English can be a digraph (*cathode*) or two separate letters (*cathouse*); *gn* in French can be a digraph (*oignon*) or two separate letters (*gnome*).

The sequence <0D31, 0D31> is rendered as ൠ, regardless of the reading of that text. The sequence <0D31, 0D4D, 0D31> is rendered as ൨. In both cases, vowels signs are applied to each rendered base, as shown in *Table 12-38*.

Table 12-38. Malayalam /ṟaṟa/ and /ṭṭa/

പാററ	0D2A 0D3E 0D31 0D31	/pāṭṭa/	cockroach
പാറ്റ	0D2A 0D3E 0D31 0D4D 0D31		
മാറെറാലി	0D2E 0D3E 0D31 0D46 0D31 0D3E 0D32 0D3F	/māṭṭoli/	echo
മാറ്റൊലി	0D2E 0D3E 0D31 0D4D 0D31 0D4A 0D32 0D3F		
ബാറററി	0D2C 0D3E 0D31 0D31 0D31 0D3F	/bāṭṭaṟi/	battery
ബാറ്ററി	0D2C 0D3E 0D31 0D4D 0D31 0D31 0D3F		
സൂറററ്	0D38 0D42 0D31 0D31 0D31 0D4D	/sūraṭṭ/	Surat, a town in Gujarat
സൂറ്റ്‌	0D38 0D42 0D31 0D31 0D4D 0D31 0D4D		
ടെംപററി	0D1F 0D46 0D02 0D2A 0D31 0D31 0D3F	/ṭemparaṟi/	temporary
ലെക്ചററോട്	0D32 0D46 0D15 0D4D 0D1A 0D31 0D31 0D4B 0D1F 0D4D	/lekcaraṟōṭ/	to the lecturer

A very similar situation exists for the combination of ൻ *chillu-n* and റ *rra*. When used side by side, ൻറ can be read either /ṉra/ or /ṉṭa/, while stacked ൻ is always read /ṉṭa/.

The sequence <0D7B, 0D31> is rendered as ൻറ, regardless of the reading of that text. The sequence <0D7B, 0D4D, 0D31> is rendered as ൻ. In both cases, vowels signs are applied to each rendered base, as shown in *Table 12-39*.

Table 12-39. Malayalam /ṉra/ and /ṉṭa/

ആൻേറാ	0D06 0D7B 0D47 0D31 0D3E	/āṉṭō/	a proper name
ആൻ്റോ	0D06 0D7B 0D4D 0D31 0D4B		
എൻറോൾ	0D0E 0D7B 0D31 0D4B 0D7E	/eṉṟōl/	enroll

Legacy Representations of Conjunct /ṉṭa/. Prior to Unicode 5.1 when <0D7B *chillu-n*, 0D4D *virama*, 0D31 *rra*> became the recommendation for the conjunct ൻ / ṉṭa/, two other representations were already in use: <0D28 *na*, 0D4D *virama*, 0D31 *rra*> and <0D28 *na*, 0D4D *virama*, 200D ZWJ, 0D31 *rra*>. All three representations are widespread because implementations have been slow to adopt the recommended representation.

Implementations should treat <*na*, *virama*, *rra*> in existing text as equivalent to the recommended representation for the conjunct ൻ്റ, <*chillu-n*, *virama*, *rra*>. Newly generated text should only use the recommended representation.

The other legacy representation <*na*, *virama*, ZWJ, *rra*> conflicts with the legacy representation of the side-by-side form ൻറ (see "Legacy Chillu Sequences" later in this section). Therefore, implementations should treat <*na*, *virama*, ZWJ, *rra*> as a representation of the stacked form ൻ്റ only if they know this sequence is not used to represent the side-by-side form ൻറ.

Legacy Chillu Sequences. Prior to Unicode Version 5.1, the representation of text with *chillu* forms was problematic, and not clearly described in the text of the standard. Because older data will use different representation for *chillu* forms, implementations must be prepared to handle both kinds of data. For *chillu* forms considered in isolation, the following table shows the relationship between their representation in Version 5.0 and earlier, and the recommended representation starting with Version 5.1. Note that only the five *chillu* forms listed in <u>Table 12-40</u> were specified in the standard before Version 5.1, and thus were represented in legacy text by <virama, ZWJ> sequences. Other *chillu* forms in Malayalam are only represented as atomically encoded *chillu* characters.

Table 12-40. Legacy Encoding of Malayalam *Chillus*

Visual	Legacy Representation (5.0)	Preferred Representation
ൺ	*nna*, *virama*, ZWJ 0D23, 0D4D, 200D	0D7A MALAYALAM LETTER CHILLU NN
ൻ	*na*, *virama*, ZWJ 0D28, 0D4D, 200D	0D7B MALAYALAM LETTER CHILLU N
ർ	*ra*, *virama*, ZWJ 0D30, 0D4D, 200D	0D7C MALAYALAM LETTER CHILLU RR
ൽ	*la*, *virama*, ZWJ 0D32, 0D4D, 200D	0D7D MALAYALAM LETTER CHILLU L
ൾ	*lla*, *virama*, ZWJ 0D33, 0D4D, 200D	0D7E MALAYALAM LETTER CHILLU LL

12.9.4 Malayalam Numbers and Punctuation

Archaic Numbers. The archaic numbering system for Malayalam included numbers for 10, 100, and 1000, as well as signs for fractions. Many Malayalam-specific fraction signs are encoded in the Malayalam block. Malayalam also made use of the fraction signs for one quarter, one half, and three quarters encoded in the Common Indic Number Forms block.

Date Mark. The *date mark* isused only for the day of the month in dates; it is roughly the equivalent of "th" in "June 5th." While it has been used in modern times it is not seen as much in contemporary use.

Punctuation. *Danda* and *double danda* marks as well as some other unified punctuation used with Malayalam are found in the Devanagari block; see *Section 12.1, Devanagari*.

Chapter 13

South and Central Asia-II
Other Modern Scripts

This chapter describes other modern scripts in South and Central Asia.

The Thaana script is used to write Dhivehi, the language of the Republic of Maldives, an island nation in the middle of the Indian Ocean.

Sinhala is an official script of Sri Lanka, where it is used to write the majority language, also known as Sinhala.

The Newa script, also known as Nepaalalipi in Nepal and as Newar in English-speaking countries, is a Brahmi-based script that dates to the tenth century CE. It was actively used in central Nepal until the late 18th century. Newa is presently used to write the Nepal Bhasa language, a Tibeto-Burman language spoken in the Kathmandu Valley of Nepal and in the Indian state of Sikkim.

The Mongolian script was developed as an adaption of the Old Uyghur alphabets around the beginning of the thirteenth century, during the reign of Genghis Khan. It is used in both China and Mongolia.

The Tibetan script is used for writing the Tibetan language in several countries and regions throughout the Himalayas. The approach to the encoding of Tibetan in the Unicode Standard differs from that for most Brahmi-derived scripts. Instead of using a virama-based model for consonant conjuncts, it uses a subjoined consonant model.

Limbu is a Brahmi-derived script primarily used to write the Limbu language, spoken mainly in eastern Nepal, Sikkim, and in the Darjeeling district of West Bengal. Its encoding follows a variant of the Tibetan model, making use of subjoined medial consonants, but also explicitly encoded syllable-final consonants.

Lepcha is the writing system for the Lepcha language, spoken in Sikkim and in the Darjeeling district of the West Bengal state of India. Lepcha is directly derived from the Tibetan script, but all of the letters were rotated by ninety degrees.

Meetei Mayek is used to write Meetei, a Tibeto-Burman language spoken primarily in Manipur, India. Like Limbu, it makes use of explicitly encoded syllable-final consonants.

Chakma is used to write the language of the Chakma people of southeastern Bangladesh and surrounding areas. The language, spoken by about half a million people, is related to other eastern Indo-European languages such as Bengali.

Saurashtra is used to write the Saurashtra language, related to Gujarati, but spoken in southern India. The Saurashtra language is most often written using the Tamil script, instead.

Ol Chiki is an alphabetic script invented in the 20th century to write Santali, a Munda language of India. It is used primarily for the southern dialect of Santali spoken in the state of Odisha (Orissa).

Ol Onal is another alphabetic script invented in the 20th century to write Bhumij, a Munda language. It is primarily spoken in the northeast of India.

The Nag Mundari script, commonly known as "Mundari Bani," is used to write the Mundari language, spoken primarily in the states of Jharkhand, West Bengal and Odisha in India. This alphabetic script was developed starting in the 1950s and is written left to right.

Mro is a Tibeto-Burman language spoken primarily in Bangladesh. The Mro script is a left-to-right alphabet used to write the Mro language. It was invented in the 1980s and is unrelated to existing scripts.

The Warang Citi script is a recently devised left-to-right alphabet. The script is used to write the Ho language, a North Munda language which has an emergent literary tradition. The Ho people live in eastern India.

The Masaram Gondi script is an *abugida*. The script was created in 1918 to write the Gondi language, a Dravidian language spoken in central and southeastern India. While not historically related to Brahmi, its general structure is similar to that of other Brahmi-derived Indic scripts. Presently, Masaram Gondi is used in handwritten and printed materials.

Gunjala Gondi is also an *abugida* based on the Brahmi model. It is named after the village in the Adilabad district of the southern Indian state of Telegana, where manuscripts in the script were found. The script is used to write the Adilabad dialect of the Gondi language. Gunjala Gondi is taught and appears in publications today, but the dialect is more commonly written in Telugu or Devanagari.

The Wancho script is an alphabet recently devised to write the Wancho language. Wancho is a Sino-Tibetan language used mainly in the southeast of Arunachal Pradesh, as well as in Assam, Nagaland, Myanmar, and Bhutan.

Toto is a left-to-right alphabetic script recently created to write Toto, a Sino-Tibetan language spoken in West Bengal, India. The script was designed by a member of the Toto community, and officially launched in 2015.

Like Warang Citi, Wancho and Toto, Tangsa is another recently created alphabetic script. Tangsa was created in 1990 to write the Tangsa languages spoken in Arunachal Pradesh, India, and the Sagaing Region of northwest Myanmar.

Sunuwar is a script created in 1942 to write the Kiranti-Kõits language, spoken in Nepal and Sikkim. In Nepal the Sunuwar script is used as an alphabet, but in Sikkim it is used a little differently, and functions there as an *abugida*.

Gurung Khema is a Brahmi-based abugida developed in the 1990s to write Gurung, a Tibeto-Burman language. It is primarily spoken in Nepal and the state of Sikkim in India.

Kirat Rai is an abugida based on a simplified Brahmic model. It is used to write the Bantawa language, spoken in eastern Nepal and in the Indian states of Sikkim and West Bengal. It is mostly used by Bantawa speakers in India, and is recognized as the official script for Bantawa in Sikkim. Kirat Rai was developed in the 1920s and has some historical connections to the Limbu script.

13.1 Thaana

13.1.1 Thaana: U+0780–U+07BF

The Thaana script is used to write the modern Dhivehi language of the Republic of Maldives, a group of atolls in the Indian Ocean. Like the Arabic script, Thaana is written from right to left and uses vowel signs, but it is not cursive. The basic Thaana letters have been extended by a small set of dotted letters used to transcribe Arabic. The use of modified Thaana letters to write Arabic began in the middle of the 20th century. Loan words from Arabic may be written in the Arabic script, although this custom is not very prevalent today. (See *Section 9.2, Arabic*.)

While Thaana's glyphs were borrowed in part from Arabic (letters *haa* through *vaavu* were based on the Arabic-Indic digits, for example), and while vowels and *sukun* are marked with combining characters as in Arabic, Thaana is properly considered an alphabet, rather than an abjad, because writing the vowels is obligatory.

Directionality. The Thaana script is written from right to left. Conformant implementations of Thaana script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Vowels. Consonants are always written with either a vowel sign (U+07A6..U+07AF) or the null vowel sign (U+07B0 THAANA SUKUN). U+0787 THAANA LETTER ALIFU with the null vowel sign denotes a glottal stop. The placement of the Thaana vowel signs is shown in *Table 13-1*.

Table 13-1. Thaana Glyph Placement

Syllable	Display
tha	ޠ
thaa	ޠ
thi	ޢ
thee	ޢ
thu	ޠ
thoo	ޠ
the	ޠ
they	ޠ
tho	ޠ
thoa	ޠ
th	ޠ

Numerals. Both European (U+0030..U+0039) and Arabic digits (U+0660..U+0669) are used. European numbers are used more commonly and have left-to-right display

directionality in Thaana. Arabic numeric punctuation is used with digits, whether Arabic or European.

Punctuation. The Thaana script uses spaces between words. It makes use of a mixture of Arabic and European punctuation, though rules of usage are not clearly defined. Sentence-final punctuation is now generally shown with a single period (U+002E "." FULL STOP) but may also use a sequence of two periods (U+002E followed by U+002E). Phrases may be separated with a comma (usually U+060C ARABIC COMMA) or with a single period (U+002E). Colons, dashes, and double quotation marks are also used in the Thaana script. In addition, Thaana makes use of U+061F ARABIC QUESTION MARK and U+061B ARABIC SEMICOLON.

Character Names and Arrangement. The character names are based on the names used in the Republic of Maldives. The character name at U+0794, *yaa,* is found in some sources as *yaviyani*, but the former name is more common today. Characters are listed in Thaana alphabetical order from *haa* to *ttaa* for the Thaana letters, followed by the extended characters in Arabic alphabetical order from *hhaa* to *waavu*.

13.2 Sinhala

13.2.1 Sinhala: U+0D80–U+0DFF

The Sinhala script, also known as Sinhalese or Singhalese, is used to write the Sinhala language, the majority language of Sri Lanka. It is also used to write the Pali and Sanskrit languages. The script is a descendant of Brahmi and resembles the scripts of South India in form and structure.

Sinhala differs from other languages of the region in that it has a series of prenasalized stops that are distinguished from the combination of a nasal followed by a stop. In other words, both forms occur and are written differently—for example, අඬ <U+0D85, U+0DAC> *aňḍa* [aⁿḍa] "sound" versus අණ්ඩ <U+0D85, U+0DAB, U+0DCA, U+0DA9> *aṇḍa* [aɳḍa] "egg." Sinhala also has distinct signs for both a short and a long low front vowel whose sound [æ] is similar to the initial vowel in the English word "apple." The independent forms of these vowels are encoded at U+0D87 and U+0D88; the corresponding dependent forms are U+0DD0 and U+0DD1.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 13-2* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 13-2. Sinhala Vowel Letters

For	Use	Do Not Use
ආ	0D86	<0D85, 0DCF>
ඇ	0D87	<0D85, 0DD0>
ඈ	0D88	<0D85, 0DD1>
ඌ	0D8C	<0D8B, 0DDF>
ඎ	0D8E	<0D8D, 0DD8>
ඐ	0D90	<0D8F, 0DDF>
ඒ	0D92	<0D91, 0DCA>
ඓ	0D93	<0D91, 0DD9>
ඖ	0D96	<0D94, 0DDF>

Other Letters for Tamil. The Sinhala script may also be used to write Tamil. In this case, some additional combinations may be required. Some letters, such as U+0DBB ර SINHALA LETTER RAYANNA and U+0DB1 න SINHALA LETTER DANTAJA NAYANNA, may be modified by adding the equivalent of a nukta. There is, however, no nukta presently encoded in the Sinhala block.

Virama (al-lakuna) and Consonant Forms. Rendering Sinhala is similar to other Brahmic scripts in the Unicode Standard, particularly Tamil, however, consonant forms are encoded differently. Each consonant may be represented as any of the following forms:

- a live consonant
- a dead consonant with a visible U+0DCA ◌ SINHALA SIGN AL-LAKUNA
- a reduced form
- a part of a ligated conjunct
- a part of a touching conjunct

Unless combined with a U+200D ⬚ ZERO WIDTH JOINER, an *al-lakuna* is always visible and does not join consonants to form orthographic consonant clusters. For example, between a pair of live consonants, U+0DAF ද *da* and U+0DB0 ධ *dha*, an *al-lakuna* alone merely results in a dead consonant *d* followed by a live consonant *dha*, without a conjunct form:

$$ ද + ◌ + ධ \rightarrow ද්ධ $$

The sequence <◌ *al-lakuna*, ⬚> joins consonants to form orthographic consonant clusters in the style of reduced forms or ligated conjuncts. The sequence <⬚, ◌ *al-lakuna*> joins consonants to form orthographic consonant clusters in the style of touching conjuncts. The latter style is productive and should not be implemented on a case-by-case basis.

$$ ද + ◌ + ⬚ + ය \rightarrow ද්‍ය \; d.ya \; (da \text{ with reduced } ya) $$

$$ ද + ◌ + ⬚ + ධ \rightarrow ද්ධ \; d.dha \; (da \text{ and } dha \text{ ligated}) $$

$$ ද + ⬚ + ◌ + ධ \rightarrow ද‍්ධ \; d.dha \; (da \text{ and } dha \text{ touching}) $$

Note how the use of ZWJ in Sinhala differs from that of typical Indic scripts. The order of an *al-lakuna* and a ZWJ between two consonants is not related to which consonant will take a reduced form, but instead affects the style of orthographic consonant clusters.

Reduced Consonant Forms. Three reduced forms are commonly recognized: *repaya*, the above-base form of *ra* when it is the first consonant in an orthographic consonant cluster; *yansaya* and *rakaaraansaya*, the post-base form of *ya* and the below-base form of *ra*, respectively, when they follow another consonant in an orthographic consonant cluster. These three reduced forms also have Unicode named character sequences, as shown in <u>Table 13-3</u>.

Table 13-3. Sinhala Named Character Sequences

Reduced Form	Representation	Name
ඊ *repaya*	ර *ra* + ් + [ZW]	SINHALA CONSONANT SIGN REPAYA
◌ය *yansaya*	් + [ZW] + ය *ya*	SINHALA CONSONANT SIGN YANSAYA
◌ *rakaaraansaya*	් + [ZW] + ර *ra*	SINHALA CONSONANT SIGN RAKAARAANSAYA

Ligated Conjuncts. <u>Table 13-4</u> lists common ligated conjuncts formed by conjoining two consonants with the sequence <් *al-lakuna*, [ZW]>. The conjunct *j.nya* is atomically encoded as U+0DA5 ඥ SINHALA LETTER TAALUJA SANYOOGA NAAKSIKYAYA.

Table 13-4. Sinhala Ligated Conjuncts

Conjunct	Representation
ක්ව *k.va*	0D9A *ka* + ් + [ZW] + 0DC0 *va*
ක්ෂ *k.ssa*	0D9A *ka* + ් + [ZW] + 0DC2 *ssa*
ග්ධ *g.dha*	0D9C *ga* + ් + [ZW] + 0DB0 *dha*
ට්ඨ *tt.ttha*	0DA7 *tta* + ් + [ZW] + 0DA8 *ttha*
ත්ථ *t.tha*	0DAD *ta* + ් + [ZW] + 0DAE *tha*
ත්ව *t.va*	0DAD *ta* + ් + [ZW] + 0DC0 *va*
ද්ධ *d.dha*	0DAF *da* + ් + [ZW] + 0DB0 *dha*
ද්ව *d.va*	0DAF *da* + ් + [ZW] + 0DC0 *va*
න්ථ *n.tha*	0DB1 *na* + ් + [ZW] + 0DAE *tha*
න්ද *n.da*	0DB1 *na* + ් + [ZW] + 0DAF *da*
න්ධ *n.dha*	0DB1 *na* + ් + [ZW] + 0DB0 *dha*
න්ව *n.va*	0DB1 *na* + ් + [ZW] + 0DC0 *va*

Special Ligatures. Certain combinations of a base followed by a dependent sign exhibit special interaction. <u>Table 13-5</u> shows the most irregular cases, which involve a consonant letter followed by a vowel sign.

Table 13-5. Irregular Vowel Sign Ligatures of Sinhala

Combination	Ligature
ර + ◌ැ	රැ *rae*
ර + ◌ෑ	රෑ *raae*
ර + ◌ු	රු *ru*
ර + ◌ූ	රූ *ruu*
ළ + ◌ු	ළු *llu*
ළ + ◌ූ	ළූ *lluu*

There are several other notable, more regular cases. Bases with a structure similar to ද *da* lose the descending tail when a below-base sign is attached.

ළ *da* + ◌ *-u* / ◌ *-uu* / ◌ *rakaaraansaya* → ළු *du*, ළූ *duu*, ළ්‍ර *d.ra*

The vowel killer *al-lakuna* takes an alternative form when combined with certain bases. For example:

ච *ca* + ◌ *al-lakuna* → ච් *c*

Vowel signs *u* and *uu* take alternative forms when combined with the following specific bases: ක *ka*, ග *ga*, ඟ *ngga*, ත *ta*, භ *bha*, and ශ *sha*. They take another pair of alternative forms when a ◌ *rakaaraansaya* is already present.

ක *ka* + ◌ *-u* / ◌ *-uu* → කු *ku*, කූ *kuu*

ක්‍ර *k.ra* + ◌ *-u* / ◌ *-uu* → ක්‍රු *k.ru*, ක්‍රූ *k.ruu*

In particular, the alternative forms shown in the second line of this example should not be encoded using the vowel signs ◌ැ *ae* and ◌ෑ *aae*, which look similar, but are rendered higher: ක්‍රැ *k.rae*, ක්‍රෑ *k.raae*.

Candrabindu for Sanskrit. U+0D81 ◌ SINHALA SIGN CANDRABINDU represents the candrabindu used in some archaic Sanskrit texts. It is not for use in modern Sinhala.

Punctuation. Sinhala currently uses Western-style punctuation marks. U+0DF4 ෴ SINHALA PUNCTUATION KUNDDALIYA was used historically as a typographic ornament or to punctuate sentences, sections, and chapters. U+0964 । DEVANAGARI DANDA is used to represent dandas that only occasionally occur in historic Sanskrit or Pali texts written in the Sinhala script.

Digits. Modern Sinhala text uses Western digits. The set of digits in the range U+0DE6 to U+0DEF was used into the twentieth century, primarily to write horoscopes. That set of astrological digits is known as Sinhala Lith Illakkam, and includes a form for zero.

Standards. SLS 1134, *Sinhala Character Code for Information Interchange*, is Sri Lanka's national standard for encoding the Sinhala script.

13.2.2 Sinhala Archaic Numbers: U+111E0–U+111FF

The Sinhala Archaic Numbers block contains characters used in a historic number system called Sinhala Illakkam, which was in use prior to 1815. Sinhala Illakkam was not a positional notation, and lacks a digit for zero. It is distinct from the set of Sinhala astrological digits called Sinhala Lith Illakkam (U+0DE6..U+0DEF).

13.3 Newa

13.3.1 Newa: U+11400–U+1147F

The Newa script, also known as Nepaalalipi in Nepal and as Newar in English-speaking countries, is a Brahmi-based script that dates to the tenth century CE. The script is attested in inscriptions, coins, manuscripts, books, and other publications.

Newa was actively used in central Nepal until the latter half of the 18th century, when the Newa dynasties were overthrown and the use of the script began to decline. In 1905 the script was banned, but the ban was lifted in 1951. Today Newa is used to write the Nepal Bhasa language, a Tibeto-Burman language spoken predominantly in the Kathmandu Valley of Nepal and in the Indian state of Sikkim. It also is used to write Sanskrit and Nepali. Historically, Newa has been used for Maithili, Bengali, and Hindi. At present, the Nepal Bhasa language is most often written in the Devanagari script.

Structure. Like other Brahmi-derived Indic scripts, Newa is an *abugida* and makes use of a virama. The script is written from left to right.

Vowels. Vowel length is usually indicated by the dependent vowel signs. The *visarga* may also be used to show vowel length. Some vowels are used only for Sanskrit and are not needed for the representation of Nepal Bhasa.

Virama and Conjuncts. Conjunct forms of consonant clusters are represented with U+11442 NEWA SIGN VIRAMA. Half-forms are used for writing horizontal conjuncts, and generally used only for consonants with right descenders. Explicit half-forms can be produced by writing U+200D ZERO WIDTH JOINER after the *virama*. In particular, a cluster-initial *ra* is rendered in its above-base form (*repha*) by default, while its half-form (eyelash) needs to be explicitly requested. Vertical conjuncts are currently preferred for writing consonant clusters, but manuscripts show more variation, such as conjuncts in a horizontal or cascading shape.

Murmured Resonant Consonants. Six consonant letters are encoded to represent murmured resonants in the Nepal Bhasa language, as shown in *Table 13-6*. The murmured resonants are analyzed as individual letters in the modern orthography, and are separately encoded. Similar-appearing conjuncts involving the consonant *ha* in Sanskrit text should be represented as conjuncts, using a sequence of <C1, virama, C2>, consistent with Sanskrit practice for other Indic scripts.

Table 13-6. Murmured Resonants in Nepal Bhasa

Code Point	Glyph	Name
U+11413	ᐎ	NEWA LETTER NGHA

U+11419	ह	NEWA LETTER NYHA
U+11424	ह	NEWA LETTER NHA
U+1142A	ह	NEWA LETTER MHA
U+1142D	ह	NEWA LETTER RHA
U+1142F	ह	NEWA LETTER LHA

Jihvamuliya and Upadhmaniya. The voiceless velar and bilabial fricatives in Newa are represented by U+11460 NEWA SIGN JIHVAMULIYA and U+11461 NEWA SIGN UPADHMANIYA respectively. These two characters have the Indic_Syllabic_Category value of Consonant_With_Stacker, which means that they make stacked ligatures with the next consonant without the use of a *virama*. Dependent vowel signs can also be added to the stack.

Rendering. Combinations of certain consonants and vowel signs may have special rendering requirements. For example, the vowel signs *ai*, *o*, and *au* have two-part contextual forms when the vowels occur after the consonants *ga*, *nya*, *ttha*, *nna*, *tha*, *dha*, and *sha*. In addition, several consonant letters have glyphic variants. These include *ga*, *jha*, *nya*, *ra*, and *sha*.

Ligatures. The consonant clusters *kṣa* and *jña* are represented by the sequences <U+1140E NEWA LETTER KA, U+11442 NEWA SIGN VIRAMA, U+11432 NEWA LETTER SSA> and <U+11416 NEWA LETTER JA, U+11442 NEWA SIGN VIRAMA, U+11423 NEWA LETTER NA>, respectively. The consonants *ja* and *ra* are also written as ligatures when combined with U+11438 NEWA VOWEL SIGN U or U+11439 NEWA VOWEL SIGN UU.

Digits. Newa has a full set of decimal digits located at U+11450 to U+11459.

Punctuation. Newa makes use of script-specific *dandas*, U+1144B NEWA DANDA and U+1144C NEWA DOUBLE DANDA. Other Newa punctuation marks include U+1144D NEWA COMMA and U+1145A NEWA DOUBLE COMMA, which are used as phrase separators, and U+1145D NEWA INSERTION SIGN. The punctuation mark U+1144E NEWA GAP FILLER indicates a break or fills a gap in a line at the margin. The character U+1145B NEWA PLACEHOLDER MARK also is used to fill a gap in a line, but may be used to mark the end of text. U+1144F NEWA ABBREVIATION SIGN is employed to indicate an abbreviation.

Unicode characters in other blocks may be used for other punctuation marks that occur in Newa texts . A flower mark, used to identify the end of a text section, can be represented by U+2055 FLOWER PUNCTUATION MARK. This mark typically occurs with U+1144C NEWA DOUBLE DANDA on either side. To indicate a deletion, U+1DFB COMBINING DELETION MARK can be used.

Other Signs. To indicate nasalization, U+11443 NEWA SIGN CANDRABINDU and U+11444 NEWA SIGN ANUSVARA are used. U+11445 NEWA SIGN VISARGA represents

post-vocalic aspiration or can be used to mark vowel length. U+11446 NEWA SIGN NUKTA is used to indicate sounds for which distinct characters in Newa do not exist, such as in loanwords. The character U+11447 NEWA SIGN AVAGRAHA marks elision of a word-initial *a* in Sanskrit as the result of *sandhi*. U+11448 NEWA SIGN FINAL ANUSVARA has different uses. In certain manuscripts, it indicates nasalization, whereas in other sources, it is a form of punctuation, similar to a semicolon.

Newa includes two invocation signs, U+11449 NEWA OM and U+1144A NEWA SIDDHI. The sign *om* may also be written using the sequence <U+1140C NEWA LETTER O, U+11443 NEWA SIGN CANDRABINDU>. The Newa sign *siddhi* is written at the beginning of text, often beside *om*. It represents *siddhirastu*, "may there be success."

13.4 Tibetan

13.4.1 Tibetan: U+0F00–U+0FFF

The Tibetan script is used for writing Tibetan in several countries and regions throughout the Himalayas. Aside from Tibet itself, the script is used in Ladakh, Nepal, and northern areas of India bordering Tibet where large Tibetan-speaking populations now reside. The Tibetan script is also used in Bhutan to write Dzongkha, the official language of that country. In Bhutan, as well as in some scholarly traditions, the Tibetan script is called the Bodhi script, and the particular version written in Bhutan is known as Joyi (mgyogs yig). In addition, Tibetan is used as the language of philosophy and liturgy by Buddhist traditions spread from Tibet into the Mongolian cultural area that encompasses Mongolia, Buriatia, Kalmykia, and Tuva.

The Tibetan scripting and grammatical systems were originally defined together in the sixth century by royal decree when the Tibetan King Songtsen Gampo sent 16 men to India to study Indian languages. One of those men, Thumi Sambhota, is credited with creating the Tibetan writing system upon his return, having studied various Indic scripts and grammars. The king's primary purpose was to bring Buddhism from India to Tibet. The new script system was therefore designed with compatibility extensions for Indic (principally Sanskrit) transliteration so that Buddhist texts could be represented properly. Because of this origin, over the last 1,500 years the Tibetan script has been widely used to represent Indic words, a number of which have been adopted into the Tibetan language retaining their original spelling.

A note on Latin transliteration: Tibetan spelling is traditional and does not generally reflect modern pronunciation. Throughout this section, Tibetan words are represented in italics when transcribed as spoken, followed at first occurrence by a parenthetical transliteration; in these transliterations, the presence of the *tsek* (tsheg)character is expressed with a hyphen.

Thumi Sambhota's original grammar treatise defined two script styles. The first, called *uchen* (dbu-can, "with head"), is a formal "inscriptional capitals" style said to be based on an old form of Devanagari. It is the script used in Tibetan xylograph books and the one used in the coding tables. The second style, called *u-mey* (dbu-med, or "headless"), is more cursive and said to be based on the Wartu script. Numerous styles of *u-mey* have evolved since then, including both formal calligraphic styles used in manuscripts and running handwriting styles. All Tibetan scripts follow the same lettering rules, though there is a slight difference in the way that certain compound stacks are formed in *uchen* and *u-mey*.

General Principles of the Tibetan Script. Tibetan grammar divides letters into consonants and vowels. There are 30 consonants, and each consonant is represented by a discrete written character. There are five vowel sounds, only four of which are

represented by written marks. The four vowels that are explicitly represented in writing are each represented with a single mark that is applied above or below a consonant to indicate the application of that vowel to that consonant. The absence of one of the four marks implies that the first vowel sound (like a short "ah" in English) is present and is not modified to one of the four other possibilities. Three of the four marks are written above the consonants; one is written below.

Each word in Tibetan has a base or root consonant. The base consonant can be written singly or it can have other consonants added above or below it to make a vertically "stacked" letter. Tibetan grammar contains a very complete set of rules regarding letter gender, and these rules dictate which letters can be written in adjacent positions. The rules therefore dictate which combinations of consonants can be joined to make stacks. Any combination not allowed by the gender rules does not occur in native Tibetan words. However, when transcribing other languages (for example, Sanskrit, Chinese) into Tibetan, these rules do not operate. In certain instances other than transliteration, any consonant may be combined with any other subjoined consonant. Implementations should therefore be prepared to accept and display any combinations.

For example, the syllable *spyir* "general," pronounced [tʃiː], is a typical example of a Tibetan syllable that includes a stack comprising a head letter, two subscript letters, and a vowel sign. *Figure 13-1* shows the characters in the order in which they appear in the backing store.

Figure 13-1. Tibetan Syllable Structure

① U+0F66 TIBETAN LETTER SA
② U+0FA4 TIBETAN SUBJOINED LETTER PA
③ U+0FB1 TIBETAN SUBJOINED LETTER YA
④ U+0F72 TIBETAN VOWEL SIGN I
⑤ U+0F62 TIBETAN LETTER RA
⑥ U+0F0B TIBETAN MARK INTERSYLLABIC TSHEG
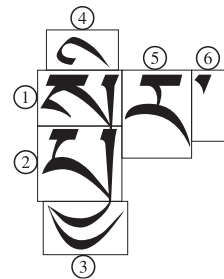
The model adopted to encode the Tibetan lettering set described above contains the following groups of items: Tibetan consonants, vowels, numerals, punctuation, ornamental signs and marks, and Tibetan-transliterated Sanskrit consonants and vowels. Each of these will be described in this section.

Both in this description and in Tibetan, the terms "subjoined" (-btags) and "head" (-mgo) are used in different senses. In the structural sense, they indicate specific slots defined in native Tibetan orthography. In spatial terms, they refer to the position in the stack; anything in the topmost position is "head," anything not in the topmost position is "subjoined." Unless explicitly qualified, the terms "subjoined" and "head"

are used here in their spatial sense. For example, in a conjunct like "rka," the letter in the root slot is "KA." Because it is not the topmost letter of the stack, however, it is expressed with a subjoined character code, while "RA", which is structurally in the head slot, is expressed with a nominal character code. In a conjunct "kra," in which the root slot is also occupied with "KA", the "KA" is encoded with a nominal character code because it is in the topmost position in the stack.

The Tibetan script has its own system of formatting, and details of that system relevant to the characters encoded in this standard are explained herein. However, an increasing number of publications in Tibetan do not strictly adhere to this original formatting system. This change is due to the partial move from publishing on long, horizontal, loose-leaf folios, to publishing in vertically oriented, bound books. The Tibetan script also has a punctuation set designed to meet needs quite different from the punctuation that has evolved for Western scripts. With the appearance of Tibetan newspapers, magazines, school textbooks, and Western-style reference books in the last 20 or 30 years, Tibetans have begun using things like columns, indented blocks of text, Western-style headings, and footnotes. Some Western punctuation marks, including brackets, parentheses, and quotation marks, are becoming commonplace in these kinds of publication. With the introduction of more sophisticated electronic publishing systems, there is also a renaissance in the publication of voluminous religious and philosophical works in the traditional horizontal, loose-leaf format—many set in digital typefaces closely conforming to the proportions of traditional hand-lettered text.

Consonants. The system described here has been devised to encode the Tibetan system of writing consonants in both single and stacked forms.

All of the consonants are encoded a first time from U+0F40 through U+0F69. There are the basic Tibetan consonants and, in addition, six compound consonants used to represent the Indic consonants *gha*, *jha*, *d.ha*, *dha*, *bha,* and *ksh.a*. These codes are used to represent occurrences of either a stand-alone consonant or a consonant in the head position of a vertical stack. Glyphs generated from these codes will always sit in the normal position starting at and dropping down from the design baseline. All of the consonants are then encoded a second time. These second encodings from U+0F90 through U+0FB9 represent consonants in subjoined stack position.

To represent a single consonant in a text stream, one of the first "nominal" set of codes is placed. To represent a stack of consonants in the text stream, a "nominal" consonant code is followed directly by one or more of the subjoined consonant codes. The stack so formed continues for as long as subjoined consonant codes are contiguously placed.

This encoding method was chosen over an alternative method that would have involved a virama-based encoding, such as Devanagari. There were two main reasons for this choice. First, the virama is not normally used in the Tibetan writing system to

create letter combinations. There is a virama in the Tibetan script, but only because of the need to represent Devanagari; called "srog-med", it is encoded at U+0F84 TIBETAN MARK HALANTA. The virama is never used in writing Tibetan words and can be—but almost never is—used as a substitute for stacking in writing Sanskrit mantras in the Tibetan script. Second, there is a prevalence of stacking in native Tibetan, and the model chosen specifically results in decreased data storage requirements. Furthermore, in languages other than Tibetan, there are many cases where stacks occur that do not appear in Tibetan-language texts; it is thus imperative to have a model that allows for any consonant to be stacked with any subjoined consonant(s). Thus a model for stack building was chosen that follows the Tibetan approach to creating letter combinations, but is not limited to a specific set of the possible combinations.

Vowels. Each of the four basic Tibetan vowel marks is coded as a separate entity. These code points are U+0F72, U+0F74, U+0F7A, and U+0F7C. For compatibility, a set of several compound vowels for Sanskrit transcription is also provided in the other code points between U+0F71 and U+0F7D. Most Tibetan users do not view these compound vowels as single characters, and their use is limited to Sanskrit words. It is acceptable for users to enter these compounds as a series of simpler elements and have software render them appropriately. Canonical equivalences are specified for all of these compound vowels, with the exception of U+0F77 TIBETAN VOWEL SIGN VOCALIC RR and U+0F79 TIBETAN VOWEL SIGN VOCALIC LL, which for historic reasons have only compatibility equivalences specified. These last two characters are deprecated, and their use is strongly discouraged.

A vowel sign may be applied either to a stand-alone consonant or to a stack of consonants. The vowel sign occurs in logical order after the consonant (or stack of consonants). Each of the vowel signs is a nonspacing combining mark. The four basic vowel marks are rendered either above or below the consonant. The compound vowel marks also appear either above or below the consonant, but in some cases have one part displayed above and one part displayed below the consonant.

All of the symbols and punctuation marks have straightforward encodings. Further information about many of them appears later in this section.

Coding Order. In general, the correct coding order for a stream of text will be the same as the order in which Tibetans spell and in which the characters of the text would be written by hand. For example, the correct coding order for the most complex Tibetan stack would be

head position consonant

first subjoined consonant

... (intermediate subjoined consonants, if any)

last subjoined consonant

subjoined vowel a-chung (U+0F71)

standard or compound vowel sign, or virama

Where used, the character U+0F39 TIBETAN MARK TSA -PHRU occurs immediately after the consonant it modifies.

Allographical Considerations. When consonants are combined to form a stack, one of them retains the status of being the principal consonant in the stack. The principal consonant always retains its stand-alone form. However, consonants placed in the "head" and "subjoined" positions to the main consonant sometimes retain their stand-alone forms and sometimes are given new, special forms. Because of this fact, certain consonants are given a further, special encoding treatment—namely, "wa" (U+0F5D), "ya" (U+0F61), and "ra" (U+0F62).

Head Position "ra". When the consonant "ra" is written in the "head" position (ra-mgo, pronounced *ra-go*) at the top of a stack in the normal Tibetan-defined lettering set, the shape of the consonant can change. It can either be a full-form shape or the full-form shape but with the bottom stroke removed (looking like a short-stemmed letter "T"). This requirement of "ra" in the head position where the glyph representing it can change shape is correctly coded by using the stand-alone "ra" consonant (U+0F62) followed by the appropriate subjoined consonant(s). For example, in the normal Tibetan ra-mgo combinations, the "ra" in the head position is mostly written as the half-ra but in the case of "ra + subjoined nya" must be written as the full-form "ra". Thus the normal Tibetan ra-mgo combinations are correctly encoded with the normal "ra" consonant (U+0F62) because it can change shape as required. It is the responsibility of the font developer to provide the correct glyphs for representing the characters where the "ra" in the head position will change shape—for example, as in "ra + subjoined nya".

Full-Form "ra" in Head Position. Some instances of "ra" in the head position require that the consonant be represented as a full-form "ra" that never changes. This is *not* standard usage for the Tibetan language itself, but rather occurs in transliteration and transcription. Only in these cases should the character U+0F6A TIBETAN LETTER FIXED-FORM RA be used instead of U+0F62 TIBETAN LETTER RA. This "ra" will always be represented as a full-form "ra consonant" and will never change shape to the form where the lower stroke has been cut off. For example, the letter combination "ra + ya", when appearing in transliterated Sanskrit works, is correctly written with a full-form "ra" followed by either a modified subjoined "ya" form or a full-form subjoined "ya" form. Note that the fixed-form "ra" should be used *only* in combinations where "ra" would normally transform into a short form but the user specifically wants to prevent that change. For example, the combination "ra + subjoined nya" never requires the use of fixed-form "ra", because "ra" normally retains its full glyph

form over "nya". It is the responsibility of the font developer to provide the appropriate glyphs to represent the encodings.

Subjoined Position "wa", "ya", and "ra". All three of these consonants can be written in subjoined position to the main consonant according to normal Tibetan grammar. In this position, *all* of them change to a new shape. The "wa" consonant when written in subjoined position is not a full "wa" letter any longer but is literally the bottom-right corner of the "wa" letter cut off and appended below it. For that reason, it is called a *wa-zur* (wa-zur or "corner of a wa") or, less frequently but just as validly, *wa-ta* (wa-btags) to indicate that it is a subjoined "wa". The consonants "ya" and "ra" when in the subjoined position are called *ya-ta* (ya-btags) and *ra-ta* (ra-btags), respectively. To encode these subjoined consonants that follow the rules of normal Tibetan grammar, the shape-changed, subjoined forms U+0FAD TIBETAN SUBJOINED LETTER WA, U+0FB1 TIBETAN SUBJOINED LETTER YA, and U+0FB2 TIBETAN SUBJOINED LETTER RA should be used.

All three of these subjoined consonants also have full-form non-shape-changing counterparts for the needs of transliterated and transcribed text. For this purpose, the full sub-joined consonants that do not change shape (encoded at U+0FBA, U+0FBB, and U+0FBC, respectively) are used where necessary. The combinations of "ra + ya" are a good example because they include instances of "ra" taking a short (ya-btags) form and "ra" taking a full-form subjoined "ya".

U+0FB0 TIBETAN SUBJOINED LETTER -A (*a-chung*) should be used only in the very rare cases where a full-sized subjoined 'a-chung letter is required. The small vowel lengthening 'a-chung encoded as U+0F71 TIBETAN VOWEL SIGN AA is *far* more frequently used in Tibetan text, and it is therefore recommended that implementations treat this character (rather than U+0FB0) as the normal subjoined 'a-chung.

Halanta (Srog-Med). Because two sets of consonants are encoded for Tibetan, with the second set providing explicit ligature formation, there is no need for a "dead character" in Tibetan. When a *halanta* (srog-med) is used in Tibetan, its purpose is to suppress the inherent vowel "a". If anything, the *halanta* should *prevent* any vowel or consonant from forming a ligature with the consonant preceding the *halanta*. In Tibetan text, this character should be displayed beneath the base character as a combining glyph and not used as a (purposeless) dead character.

Line Breaking Considerations. Tibetan text separates units called natively *tsek-bar* ("tsheg-bar"), an inexact translation of which is "syllable." *Tsek-bar* is literally the unit of text between *tseks* and is generally a consonant cluster with all of its prefixes, suffixes, and vowel signs. It is not a "syllable" in the English sense.

Tibetan script has only two break characters. The primary break character is the standard interword *tsek* (tsheg), which is encoded at U+0F0B. The second break char-

acter is the space. Space or *tsek* characters in a stream of Tibetan text are not always break characters and so need proper contextual handling.

The primary delimiter character in Tibetan text is the *tsek* (U+0F0B TIBETAN MARK INTERSYLLABIC TSHEG). In general, automatic line breaking processes may break after any occurrence of this *tsek*, except where it follows a U+0F44 TIBETAN LETTER NGA (with or without a vowel sign) and precedes a *shay* (U+0F0D), or where Tibetan grammatical rules do not permit a break. (Normally, *tsek* is not written before *shay* except after "nga". This type of tsek-after-nga is called "nga-phye-tsheg" and may be expressed by U+0F0B or by the special character U+0F0C, a nonbreaking form of *tsek*.) The Unicode names for these two types of *tsek* are misnomers, retained for compatibility. The standard *tsek* U+0F0B TIBETAN MARK INTERSYLLABIC TSHEG is always required to be a potentially breaking character, whereas the "nga-phye-tsheg" is always required to be a nonbreaking *tsek*. U+0F0C TIBETAN MARK DELIMITER TSHEG BSTAR is specifically not a "delimiter" and is not for general use.

There are no other break characters in Tibetan text. Unlike English, Tibetan has no system for hyphenating or otherwise breaking a word within the group of letters making up the word. Tibetan text formatting does not allow text to be broken within a word.

Whitespace appears in Tibetan text, although it should be represented by U+00A0 NO-BREAK SPACE instead of U+0020 SPACE. Tibetan text breaks lines after *tsek* instead of at whitespace.

Complete Tibetan text formatting is best handled by a formatter in the application and not just by the code stream. If the interword and nonbreaking *tseks* are properly employed as breaking and nonbreaking characters, respectively, and if all spaces are nonbreaking spaces, then any application will still wrap lines correctly on that basis, even though the breaks might be sometimes inelegant.

Tibetan Punctuation. The punctuation apparatus of Tibetan is relatively limited. The principal punctuation characters are the *tsek*; the *shay* (transliterated "shad"), which is a vertical stroke used to mark the end of a section of text; the space used sparingly as a space; and two of several variant forms of the *shay* that are used in specialized situations requiring a *shay*. There are also several other marks and signs but they are sparingly used.

The *shay* at U+0F0D marks the end of a piece of text called "tshig-grub". The mode of marking bears no commonality with English phrases or sentences and should not be described as a delimiter of phrases. In Tibetan grammatical terms, a *shay* is used to mark the end of an expression ("brjod-pa") and a complete expression. Two *shays* are used at the end of whole topics ("don-tshan"). Because some writers use the double *shay* with a different spacing than would be obtained by coding two adjacent occurrences of U+0F0D, the double *shay* has been coded at U+0F0E with the intent

that it would have a larger spacing between component *shays* than if two *shays* were simply written together. However, most writers do not use an unusual spacing between the double *shay*, so the application should allow the user to write two U+0F0D codes one after the other. Additionally, font designers will have to decide whether to implement these *shays* with a larger than normal gap.

The U+0F11 *rin-chen-pung-shay* (rin-chen-spungs-shad) is a variant *shay* used in a specific "new-line" situation. Its use was not defined in the original grammars but Tibetan tradition gives it a highly defined use. The *drul-shay* ("sbrul-shad") is likewise not defined by the original grammars but has a highly defined use; it is used for separating sections of meaning that are equivalent to topics ("don-tshan") and subtopics. A *drul-shay* is usually surrounded on both sides by the equivalent of about three spaces (though no rule is specified). Hard spaces will be needed for these instances because the *drul-shay* should not appear at the beginning of a new line and the whole structure of spacing-plus-*shay* should not be broken up, if possible.

Tibetan texts use a *yig-go* ("head mark," yig-mgo) to indicate the beginning of the front of a folio, there being no other certain way, in the loose-leaf style of traditional Tibetan books, to tell which is the front of a page. The head mark can and does vary from text to text; there are many different ways to write it. The common type of head mark has been provided for with U+0F04 TIBETAN MARK INITIAL YIG MGO MDUN MA and its extension U+0F05 TIBETAN MARK CLOSING YIG MGO SGAB MA. An initial mark *yig-go* can be written alone or combined with as many as three closing marks following it. When the initial mark is written in combination with one or more closing marks, the individual parts of the whole must stay in proper registration with each other to appear authentic. Therefore, it is strongly recommended that font developers create precomposed ligature glyphs to represent the various combinations of these two characters. The less common head marks mainly appear in Nyingmapa and Bonpo literature. Three of these head marks have been provided for with U+0F01, U+0F02, and U+0F03; however, many others have not been encoded. Font developers will have to deal with the fact that many types of head marks in use in this literature have not been encoded, cannot be represented by a replacement that has been encoded, and will be required by some users.

Two characters, U+0F3C TIBETAN MARK ANG KHANG GYON and U+0F3D TIBETAN MARK ANG KHANG GYAS, are paired punctuation; they are typically used together to form a roof over one or more digits or words. In this case, kerning or special ligatures may be required for proper rendering. The right *ang khang* may also be used much as a single closing parenthesis is used in forming lists; again, special kerning may be required for proper rendering. The marks U+0F3E TIBETAN SIGN YAR TSHES and U+0F3F TIBETAN SIGN MAR TSHES are paired signs used to combine with digits; special glyphs or compositional metrics are required for their use.

A set of frequently occurring astrological and religious signs specific to Tibetan is encoded between U+0FBE and U+0FCF.

U+0F34, which means "et cetera" or "and so on," is used after the first few *tsek-bar* of a recurring phrase. U+0FBE (often three times) indicates a refrain.

U+0F36 and U+0FBF are used to indicate where text should be inserted within other text or as references to footnotes or marginal notes.

Svasti Signs. The *svasti* signs encoded in the range U+0FD5..U+0FD8 are widely used sacred symbols associated with Hinduism, Buddhism, and Jainism. They are often printed in religious texts, marriage invitations, and decorations, and are considered symbols of good luck and well-being. In the Hindu tradition in India, the dotted forms are often used. The *svasti* signs are used to mark religious flags in Jainism and also appear on Buddhist temples, or as map symbols to indicate the location of Buddhist temples throughout Asia. These signs are encoded in the Tibetan block, but are intended for general use; they occur with many other scripts in Asia.

In the Tibetan language, the right-facing *svasti* sign is referred to as *gyung drung nang -khor* and the left-facing *svasti* sign as *gyung drung phyi -khor*. U+0FCC TIBETAN SYMBOL NOR BU BZHI -KHYIL, or quadruple body symbol, is a Tibetan-specific version of the left-facing *svasti* sign.

The *svasti* signs have also been borrowed into the Han script and adapted as CJK ideographs. The CJK unified ideographs U+534D and U+5350 correspond to the left-facing and right-facing *svasti* signs, respectively. These CJK unified ideographs have adopted Han script-specific features and properties: they share metrics and type style characteristics with other ideographs, and are given radicals and stroke counts like those for other ideographs.

Other Characters. The Wheel of Dharma, which occurs sometimes in Tibetan texts, is encoded in the Miscellaneous Symbols block at U+2638.

The marks U+0F35 TIBETAN MARK NGAS BZUNG NYI ZLA and U+0F37 TIBETAN MARK NGAS BZUNG SGOR RTAGS conceptually attach to a *tsek-bar* rather than to an individual character and function more like attributes than characters—for example, as underlining to mark or emphasize text. In Tibetan interspersed commentaries, they may be used to tag the *tsek-bar* belonging to the root text that is being commented on. The same thing is often accomplished by setting the *tsek-bar* belonging to the root text in large type and the commentary in small type. Correct placement of these glyphs may be problematic. If they are treated as normal combining marks, they can be entered into the text following the vowel signs in a stack; if used, their presence will need to be accounted for by searching algorithms, among other things.

Tibetan Half-Numbers. The half-number forms (U+0F2A..U+0F33) are peculiar to Tibetan, though other scripts (for example, Bangla) have similar fractional concepts. The half-numbers are not well attested. Based on current evidence, the value of a half-number is 0.5 less than the number within which it appears. Half numbers

appear as the *last* digit of a multidigit number. For example, the sequence of digits "U+0F24 U+0F2C" could represent the numerical value 42.5.

Tibetan Transliteration and Transcription of Other Languages. Tibetan traditions are in place for transliterating other languages. Most commonly, Sanskrit has been the language being transliterated, although Chinese has become more common in modern times. Additionally, Mongolian has a transliterated form. There are even some conventions for transliterating English. One feature of Tibetan script/grammar is that it allows for totally accurate transliteration of Sanskrit. The basic Tibetan letterforms and punctuation marks contain most of what is needed, although a few extra things are required. With these additions, Sanskrit can be transliterated perfectly into Tibetan, and the Tibetan transliteration can be rendered backward perfectly into Sanskrit with no ambiguities or difficulties.

The six Sanskrit retroflex letters are interleaved among the other consonants.

The compound Sanskrit consonants are not used in normal Tibetan. Precomposed forms of aspirate letters (and the conjunct "kssa") are explicitly coded, along with their corresponding subjoined forms: for example, U+0F43 TIBETAN LETTER GHA, and U+0F93 TIBETAN SUBJOINED LETTER GHA, or U+0F69 TIBETAN LETTER KSSA, and U+0FB9 TIBETAN SUBJOINED LETTER KSSA. However, these characters, including the subjoined forms, decompose to stacked sequences involving subjoined "ha" (or "reversed sha") in all Unicode normalization forms.

The vowel signs of Sanskrit not included in Tibetan are encoded with other vowel signs between U+0F70 and U+0F7D. U+0F7F TIBETAN SIGN RNAM BCAD (*nam chay*) is the visarga, and U+0F7E TIBETAN SIGN RJES SU NGA RO (*nga-ro*) is the anusvara. See *Section 12.1, Devanagari*, for more information on these two characters.

The characters encoded in the range U+0F88..U+0F8B are used in transliterated text and are most commonly found in Kalachakra literature.

When the Tibetan script is used to transliterate Sanskrit, consonants are sometimes stacked in ways that are not allowed in native Tibetan stacks. Even complex forms of this stacking behavior are catered for properly by the method described earlier for coding Tibetan stacks.

Other Signs. U+0F09 TIBETAN MARK BSKUR YIG MGO is a list enumerator used at the beginning of administrative letters in Bhutan, as is the petition honorific U+0F0A TIBETAN MARK BKA- SHOG YIG MGO.

U+0F3A TIBETAN MARK GUG RTAGS GYON and U+0F3B TIBETAN MARK GUG RTAGS GYAS are paired punctuation marks (brackets).

The sign U+0F39 TIBETAN MARK TSA -PHRU (*tsa-'phru*, which is a lenition mark) is the ornamental flaglike mark that is an integral part of the three consonants U+0F59

TIBETAN LETTER TSA, U+0F5A TIBETAN LETTER TSHA, and U+0F5B TIBETAN LETTER DZA. Although those consonants are not decomposable, this mark has been abstracted and may by itself be applied to "pha" and other consonants to make new letters for use in transliteration and transcription of other languages. For example, in modern literary Tibetan, it is one of the ways used to transcribe the Chinese "fa" and "va" sounds not represented by the normal Tibetan consonants. *Tsa-'phru* is also used to represent *tsa*, *tsha*, and *dza* in abbreviations.

Traditional Text Formatting and Line Justification. Native Tibetan texts ("pecha") are written and printed using a justification system that is, strictly speaking, right-ragged but with an attempt to right-justify. Each page has a margin. That margin is usually demarcated with visible border lines required of a pecha. In modern times, when Tibetan text is produced in Western-style books, the margin lines may be dropped and an invisible margin used. When writing the text within the margins, an attempt is made to have the lines of text justified up to the right margin. To do so, writers keep an eye on the overall line length as they fill lines with text and try manually to justify to the right margin. Even then, a gap at the right margin often cannot be filled. If the gap is short, it will be left as is and the line will be said to be justified enough, even though by machine-justification standards the line is not truly flush on the right. If the gap is large, the intervening space will be filled with as many *tseks* as are required to justify the line. Again, the justification is not done perfectly in the way that English text might be perfectly right-justified; as long as the last *tsek* is more or less at the right margin, that will do. The net result is that of a right-justified, blocklike look to the text, but the actual lines are always a little right-ragged.

Justifying *tseks* are nearly always used to pad the end of a line when the preceding character is a *tsek*—in other words, when the end of a line arrives in the middle of tshig-grub (see the previous definition under "Tibetan Punctuation"). However, it is unusual for a line that ends at the end of a tshig-grub to have justifying *tseks* added to the *shay* at the end of the tshig-grub. That is, a sequence like that shown in the first line of *Figure 13-2* is not usually padded as in the second line of *Figure 13-2*, though it is allowable. In this case, instead of justifying the line with *tseks*, the space between *shays* is enlarged and/or the whitespace following the final *shay* is usually left as is. Padding is *never* applied following an actual space character. For example, given the existence of a space after a *shay*, a line such as the third line of *Figure 13-2* may not be written with the padding as shown because the final *shay* should have a space after it, and padding is never applied after spaces. The same rule applies where the final *consonant* of a tshig-grub that ends a line is a "ka" or "ga". In that case, the ending *shay* is dropped but a space is still required after the consonant and that space must not be padded. For example, the appearance shown in the fourth line of *Figure 13-2* is not acceptable.

Figure 13-2. Justifying Tibetan Tseks

Tibetan text has two rules for formatting text at the beginning of a new line. There are severe constraints on which characters can start a new line, and the first rule is traditionally stated as follows: A *shay* of any description may never start a new line. Nothing except actual words of text can start a new line, with the only exception being a *yig-go* (yig-mgo) at the head of a front page or a *da-tshe* (zla-tshe, meaning "crescent moon"—for example, U+0F05) or one of its variations, which is effectively an "in-line" *yig-go* (yig-mgo), on any other line. One of two or three ornamental *shays* is also commonly used in short pieces of prose in place of the more formal *da-tshe*. This also means that a space may not start a new line in the flow of text. If there is a major break in a text, a new line might be indented.

A syllable (tsheg-bar) that comes at the end of a tshig-grub and that starts a new line must have the *shay* that would normally follow it replaced by a rin-chen-spungs-shad (U+0F11). The reason for this second rule is that the presence of the rin-chen-spungs-shad makes the end of tshig-grub more visible and hence makes the text easier to read.

In verse, the second *shay* following the first rin-chen-spungs-shad is sometimes replaced by a rin-chen-spungs-shad, though the practice is formally incorrect. It is a writer's trick done to make a particular scribing of a text more elegant. Although a moderately popular device, it does break the rule. Not only is rin-chen-spungs-shad used as the replacement for the *shay* but a whole class of "ornamental *shays*" are used for the same purpose. All are scribal variants on a rin-chen-spungs-shad, which is correctly written with three dots above it.

Tibetan Shorthand Abbreviations (bskungs-yig) and Limitations of the Encoding. A consonant functioning as the word base (ming-gzhi) is allowed to take only one vowel sign according to Tibetan grammar. The Tibetan shorthand writing technique called bskungs-yig does allow one or more words to be contracted into a single, very unusual combination of consonants and vowels. This construction frequently entails the application of more than one vowel sign to a single consonant or stack, and the composition of the stacks themselves can break the rules of normal Tibetan grammar. For this reason, vowel signs sometimes interact typographically, which accounts for their particular combining classes (see *Section 4.3, Combining Classes*).

The Unicode Standard accounts for plain text compounds of Tibetan that contain at most one base consonant, any number of subjoined consonants, followed by any number of vowel signs. This coverage constitutes the vast majority of Tibetan text. Rarely, stacks are seen that contain more than one such consonant-vowel combination in a vertical arrangement. These stacks are highly unusual and are considered beyond the scope of plain text rendering. They may be handled by higher-level mechanisms.

13.5 Mongolian

13.5.1 Mongolian: U+1800–U+18AF

The Mongolians are key representatives of a cultural-linguistic group known as Altaic, after the Altai mountains of central Asia. In the past, these peoples have dominated the vast expanses of Asia and beyond, from the Baltic to the Sea of Japan. Echoes of Altaic languages remain from Finland, Hungary, and Turkey, across central Asia, to Korea and Japan. Today the Mongolians are represented politically in the country of Mongolia (also known as Outer Mongolia) and Inner Mongolia (formally the Inner Mongolia Autonomous Region, China), with Mongolian populations also living in other areas of China and Russia.

Implementation Guidelines. Guidelines for text representation and text shaping of the Mongolian script can be found in <u>Unicode Technical Note #57, Encoding and Shaping of the Mongolian Script</u>. For text representation, the document lists the characters and their variants for letters and marks used in each writing system; for text shaping, the document provides hierarchical shaping rules, detailing the behavior of each shaping step and the conditions for the presentation of each variant. (Unicode Technical Notes do not have normative status for the Unicode Standard.)

Unification of Characters in Mongolian Block. The Mongolian block unifies the traditional writing system for the Mongolian language and the three derivative writing systems Todo, Manchu, and Sibe. The traditional writing system is also known as "Hudum Mongolian," and is explicitly referred to as "Hudum" in the following text. Each of the three derivative writing systems shares some common letters with Hudum, and these letters are encoded only once. Each derivative writing system also has a number of modified letterforms or new letters, which are encoded separately. The letters typically required by each writing system's modern usage are encoded as shown in *Table 13-7*.

Table 13-7. Letter Usage in Mongolian Writing Systems

Hudum	Todo	Manchu	Sibe	Note
1820..1842	1820 1828 182F..1831 1834 1837..1838 1840	1820 1823 1828..182A 182E..1830 1834..1836 1838 183A	1820 1823 1828 182A 182E..1830 1834 1836..1838 183A	Unified Mongolian letters

	1843..185A 185C	185D 185F..1861 1864..1869 186C..1871 1873..1877	185D..1872	Letters specific to the derivative writing systems

Mongolian, Todo, and Manchu also have a number of special "Ali Gali" letters that are used for transcribing Tibetan and Sanskrit in Buddhist texts.

Mongolian Code Charts. The Mongolian code chart previously included glyphs for positional forms of atomic characters since Version 9.0 and standardized variation sequences since Version 7.0. As of Version 13.0, the glyphs and information about positional forms are no longer included. For a copy of the last Mongolian code chart before the Version 13.0 change, as well as the rationale behind the change, see Unicode Technical Report #54, "Unicode Mongolian 12.1 Snapshot."

History. The Mongolian script was created around the beginning of the thirteenth century, during the reign of Genghis Khan. It derives from the Old Uyghur script, which was in use from about the eighth to the fifteenth century. Old Uyghur itself was an adaptation of Sogdian Aramaic, a Semitic script written horizontally from right to left. Probably under the influence of the Chinese script, the Old Uyghur script became rotated ninety degrees counterclockwise so that the lines of text read vertically in columns running from left to right. The Mongolian script inherited this directionality from the Old Uyghur script.

The Mongolian script has remained in continuous use for writing Mongolian within the Inner Mongolia Autonomous Region of China and elsewhere in China. However, in Mongolia (Outer Mongolia), the traditional script was replaced by a Cyrillic orthography in the early 1940s. The traditional script was revived in the early 1990s, so that now both the Cyrillic and the Mongolian scripts are used. The spelling used with the traditional Mongolian script represents the literary language of the seventeenth and early eighteenth centuries, whereas the Cyrillic script is used to represent the modern, colloquial pronunciation of words. As a consequence, there is no one-to-one relationship between the traditional Mongolian orthography and Cyrillic orthography. Approximate correspondence mappings are indicated in the code charts, but are not necessarily unique in either direction. All of the Cyrillic characters needed to write Mongolian are included in the Cyrillic block of the Unicode Standard.

In addition to the traditional Mongolian script of Mongolia, several historical modifications and adaptations of the Mongolian script have emerged elsewhere. These adaptations are often referred to as scripts in their own right, although for the purposes of character encoding in the Unicode Standard they are treated as styles of the Mongolian script and share encoding of their basic letters.

The Todo script is a modified and improved version of the Mongolian script, devised in 1648 by Zaya Pandita for use by the Kalmyk Mongolians, who had migrated to

Russia in the sixteenth century, and who now inhabit the Republic of Kalmykia in the Russian Federation. The name *Todo* means "clear" in Mongolian; it refers to the fact that the new script eliminates the ambiguities inherent in the original Mongolian script. The orthography of the Todo script also reflects the Oirat-Kalmyk dialects of Mongolian rather than literary Mongolian. In Kalmykia, the Todo script was replaced by a succession of Cyrillic and Latin orthographies from the mid-1920s and is no longer in active use. Until very recently the Todo script was still used by speakers of the Oirat and Kalmyk dialects within Xinjiang and Qinghai in China.

The Manchu script is an adaptation of the Mongolian script used to write Manchu, a Tungusic language that is not closely related to Mongolian. The Mongolian script was first adapted for writing Manchu in 1599 under the orders of the Manchu leader Nurhachi, but few examples of this early form of the Manchu script survive. In 1632, the Manchu scholar Dahai reformed the script by adding circles and dots to certain letters in an effort to distinguish their different sounds and by devising new letters to represent the sounds of the Chinese language. When the Manchu people conquered China to rule as the Qing dynasty (1644–1911), Manchu become the language of state. The ensuing systematic program of translation from Chinese created a large and important corpus of books written in Manchu. Over time the Manchu people became completely sinified, and as a spoken language Manchu is now almost extinct.

The Sibe (also spelled Sibo, Xibe, or Xibo) people are closely related to the Manchus, and their language is often classified as a dialect of Manchu. The Sibe people are widely dispersed across northwest and northeast China due to deliberate programs of ethnic dispersal during the Qing dynasty. The majority have become assimilated into the local population and no longer speak the Sibe language. However, there is a substantial Sibe population in the Sibe Autonomous County in the Ili River valley in Western Xinjiang, the descendants of border guards posted to Xinjiang in 1764, who still speak and write the Sibe language. The Sibe script is based on the Manchu script, with a few modified letters.

Directionality. The Mongolian script is written vertically from top to bottom in columns running from left to right. In modern contexts, words or phrases may be embedded in horizontal scripts. In such a case, the Mongolian text will be rotated ninety degrees counterclockwise so that it reads from left to right.

When rendering Mongolian text in a system that does not support vertical layout, the text should be laid out in horizontal lines running left to right. If such text is viewed sideways, the usual Mongolian column order appears reversed, but this orientation can be workable for short stretches of text. There are no bidirectional effects in such a layout because all text is horizontal left to right.

Encoding Principles. The encoding model for Mongolian is somewhat different from that for any other script within Unicode, and in many respects it is the most

complicated. For this reason, only the essential features of Mongolian shaping behavior are presented here.

The Semitic alphabet from which the Mongolian script was ultimately derived is fundamentally inadequate for representing the sounds of the Mongolian language. As a result, many of the Mongolian letters are used to represent two different sounds, and the correct pronunciation of a letter may be known only from the context. In this respect, Mongolian orthography is similar to English spelling, in which the pronunciation of a letter such as *c* may be known only from the context.

Unlike in the Latin script, in which *c* /k/ and *c* /s/ are treated as the same letter and encoded as a single character, in the Mongolian script different phonetic values of the same glyph may be encoded as distinct characters. Modern Mongolian grammars consider the phonetic value of a letter to be its distinguishing feature, rather than its glyph shape. For example, the four Mongolian vowels *o*, *u*, *ö,* and *ü* are considered four distinct letters and are encoded as four characters (U+1823, U+1824, U+1825, and U+1826, respectively), even though *o* is written identically to *u* in all positional forms, *ö* is written identically to *ü* in all positional forms, *o* and *u* are normally distinguished from *ö* and *ü* only in the first syllable of a word. Likewise, the letters *t* (U+1832) and *d* (U+1833) are often indistinguishable. For example, pairs of Mongolian words such as *urtu* "long" and *ordu* "palace, camp, horde" or *ende* "here" and *ada* "devil" are written identically, but are represented using different sequences of Unicode characters, as shown in *Figure 13-3*. There are many such examples in Mongolian, but not in Todo, Manchu, or Sibe, which have largely eliminated ambiguous letters.

Figure 13-3. Mongolian Glyph Convergence
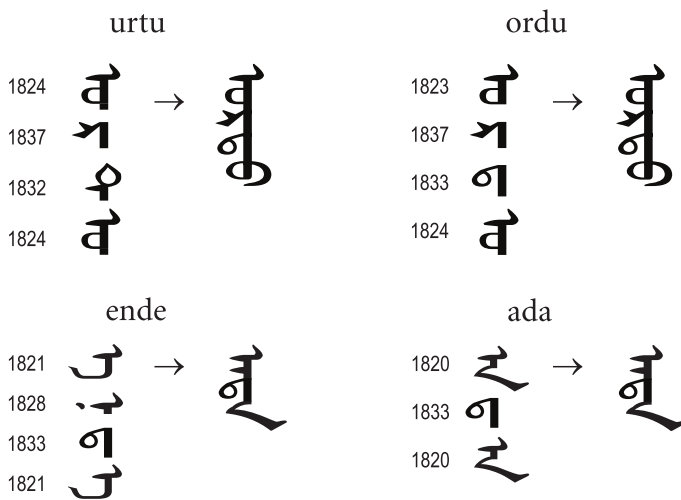
Cursive Joining. The Mongolian script is cursive, and the letters constituting a word are normally joined together. In most cases the letters join together naturally along a vertical stem, but in the case of certain "bowed" consonants (for example, U+182A MONGOLIAN LETTER BA and the feminine form of U+182C MONGOLIAN LETTER QA), which lack a trailing vertical stem, they may form ligatures with a following vowel. This is illustrated in *Figure 13-4*, where the letter *ba* combines with the letter *u* to form a ligature in the Mongolian word *abu* "father."

Figure 13-4. Mongolian Ligation
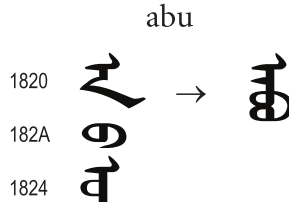
The Joining_Type values for Mongolian characters are defined in ArabicShaping.txt in the Unicode Character Database. For a discussion of the meaning of Joining_Type values in the context of a vertically rendered script, see "Cursive Joining" in *Section 14.4, Phags-pa*. Most Mongolian characters are Dual_Joining, as they may join on both top and bottom.

Many letters also have distinct glyph forms depending on their position within a word. These positional forms are classified as initial, medial, final, or isolate. The medial form is often the same as the initial form, but the final form is always distinct from the initial or medial form. *Figure 13-5* shows the Mongolian letters U+1823 *o* and U+1821 *e*, rendered with distinct positional forms initially and finally in the Mongolian words *odo* "now" and *ene* "this."

Figure 13-5. Mongolian Positional Forms
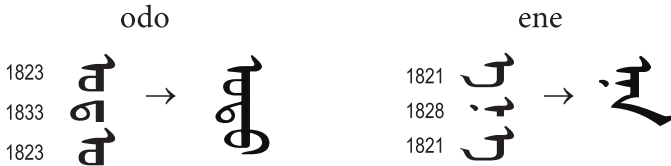
U+200C ZERO WIDTH NON-JOINER (ZWNJ) and U+200D ZERO WIDTH JOINER (ZWJ) may be used to select a particular positional form of a letter in isolation or to override the expected positional form within a word. Basically, they evoke the same contextual selection effects in neighboring letters as do non-joining or joining regular letters, but are themselves invisible (see *Chapter 23, Special Areas and Format Characters*). For example, the various positional forms of U+1820 MONGOLIAN LETTER A may be selected by means of the following character sequences:

<1820> selects the default isolate form.

<1820 200D> selects the default initial form.

<200D 1820> selects the default final form.

<200D 1820 200D> selects the default medial form.

Some letters have additional variant forms that do not depend on their position within a word, but instead reflect differences between modern versus traditional orthographic practice or lexical considerations—for example, special forms used for writing foreign words. On occasion, other contextual rules may condition a variant form selection. For example, a certain variant of a letter may be required when it occurs in the first syllable of a word or when it occurs immediately after a particular letter. The selection of the variant used for each letter is determined by hierarchical shaping steps. See Unicode Technical Note #57, Encoding and Shaping of the Mongolian Script.

The various positional and variant glyph forms of a letter are considered presentation forms and are not encoded separately. It is the responsibility of the rendering system to select the correct glyph form for a letter according to its context.

Free Variation Selectors. When a glyph form that cannot be predicted algorithmically is required (for example, when writing a foreign word), the user needs to append an appropriate variation selector to the letter to indicate to the rendering system which glyph form is required. The following free variation selectors are provided for use specifically with the Mongolian block:

U+180B MONGOLIAN FREE VARIATION SELECTOR ONE (FVS1)

U+180C MONGOLIAN FREE VARIATION SELECTOR TWO (FVS2)

U+180D MONGOLIAN FREE VARIATION SELECTOR THREE (FVS3)

U+180F MONGOLIAN FREE VARIATION SELECTOR FOUR (FVS4)

Warning

The list of standardized variants in StandardizedVariants.txt for Mongolian has not yet been updated to synchronize with the requirements of current practice as stated in Unicode Technical Note #57, Encoding and Shaping of the Mongolian Script. This defect will be addressed in a future version of the Unicode Standard.

These format characters normally have no visual appearance. When required, a free variation selector immediately follows the base character it modifies. This combination of base character and variation selector is known as a standardized variant. The table of standardized variants, StandardizedVariants.txt, in the Unicode Character Database exhaustively lists all currently defined standardized variants. All combinations not listed in the table are unspecified and are reserved for future standardization; no conformant process may interpret them as standardized variants. Therefore, any free variation selector not immediately preceded by one of their defined base characters will be ignored.

Figure 13-6 gives an example of how a free variation selector may be used to select a particular glyph variant. In modern orthography, the initial letter *ga* in the Mongolian word *gal* "fire" is written with two dots; in traditional orthography, the letter *ga* is written without any dots. By default, the dotted form of the letter *ga* is selected, but this behavior may be overridden by means of FVS1, so that *ga* plus FVS1 selects the undotted form of the letter *ga*.

Figure 13-6. Mongolian Free Variation Selector

182D 𑑑   +   1820 𑑑 + 182F 𑑑 → 𑑑

182D 𑑑 + 180B 𑑑 + 1820 𑑑 + 182F 𑑑 → 𑑑

It is important to appreciate that even though a particular standardized variant may be defined for a letter, the user needs to apply the appropriate free variation selector only if the correct glyph form cannot be predicted automatically by the rendering system. In most cases, in running text, there will be few occasions when a free variation selector is required to disambiguate the glyph form.

Older documentation, external to the Unicode Standard, listed the action of the free variation selectors by using ZWJ to explicitly indicate the shaping environment affected by the variation selector. The relative order of the ZWJ and the free variation selector in these documents was different from the one required by *Section 23.4, Variation Selectors*. Older implementations of Mongolian free variation selectors may therefore interpret a sequence such as a base character followed first by ZWJ and then by FVS1 as if it were a base character followed first by FVS1 and then by ZWJ.

Representative Glyphs. The representative glyph in the code charts is generally the isolate form for the vowels and the initial form for the consonants. Letters that share the same glyph forms are distinguished by using different positional forms for the representative glyph. For example, the representative glyph for U+1823 MONGOLIAN LETTER O is the isolate form, whereas the representative glyph for U+1824 MONGOLIAN LETTER U is the initial form. However, this distinction is only nominal, as the glyphs for the two characters are identical for the same positional form. Likewise, the

representative glyphs for U+1863 MONGOLIAN LETTER SIBE KA and U+1874 MONGO-LIAN LETTER MANCHU KA both take the final form, as their initial forms are identical to the representative glyph for U+182C MONGOLIAN LETTER QA (the initial form).

Vowel Harmony. Mongolian has a system of vowel harmony, whereby the vowels in a word are either all "masculine" and "neuter" vowels (that is, back vowels plus /i/) or all "feminine" and "neuter" vowels (that is, front vowels plus /i/). Words that are written with masculine/neuter vowels are considered to be masculine, and words that are written with feminine/neuter vowels are considered to be feminine. Words with only neuter vowels behave as feminine words (for example, take feminine suffixes). Manchu and Sibe have a similar system of vowel harmony, although it is not so strict. Some words in these two scripts may include both masculine and feminine vowels, and separated suffixes with masculine or feminine vowels may be applied to a stem irrespective of its gender.

Vowel harmony is an important element of the encoding model, as the gender of a word determines the glyph form of the velar series of consonant letters for Mongolian, Todo, Sibe, and Manchu. In each script, the velar letters have both masculine and feminine forms. For Mongolian and Todo, the masculine and feminine forms of these letters have different pronunciations.

When one of the velar consonants precedes a vowel, it takes the masculine form before masculine vowels, and the feminine form before feminine or neuter vowels. In the latter case, a ligature of the consonant and vowel is required.

When one of these consonants precedes another consonant or is the final letter in a word, it may take either a masculine or feminine glyph form, depending on its context. The rendering system should automatically select the correct gender form for these letters based on the gender of the word (in Mongolian and Todo) or the gender of the preceding vowel (in Manchu and Sibe). This is illustrated by *Figure 13-7*, where U+182D MONGOLIAN LETTER GA takes a masculine glyph form when it occurs finally in the masculine word *jarlig* "order," but takes a feminine glyph form when it occurs finally in the feminine word *cherig* "soldier." In this example, the gender form of the final letter *ga* depends on whether the first vowel in the word is a back (masculine) vowel or a front (feminine or neuter) vowel. Where the gender is ambiguous or a form not derivable from the context is required, the user needs to specify which form is required by means of the appropriate free variation selector.

Figure 13-7. Mongolian Gender Forms

1835 ᠵ + 1820 ᠠ + 1837 ᠷ + 182F ᠯ + 1822 ᠢ + 182D ᠭ →

1834 ᠴ + 1821 ᠡ +   1837 ᠷ   + 1822 ᠢ + 182D ◌ᠭ →

Special Shaping for Separated Vowels. In Mongolian, the letters *a* (U+1820) and *e* (U+1821) in a word-final position may take a "leftward tail" form or a "rightward tail" form depending on the preceding consonant that they are attached to. In some words, a final letter *a* or *e* is disconnected from the preceding consonant, in which case the vowel always takes the "leftward tail" form. U+180E MONGOLIAN VOWEL SEPARATOR (MVS) is used to represent the break between a final letter *a* or *e* and the rest of the word. MVS divides a word and disconnects the two parts, however, the *a* or *e* following the MVS is not a suffix but an integral part of the word stem.

Whether a final letter *a* or *e* is joined or separated is purely lexical and is not a question of varying orthography. This distinction is shown in <u>*Figure 13-8*</u>. The example on the left shows the word *qana* <182C, 1820, 1828, 1820> without a break before the final letter *a*, which means "the outer casing of a vein." The example on the right shows the word *qana* <182C, 1820, 1828, 180E, 1820> with a break before the final letter *a*, which means "the wall of a tent."

Figure 13-8. Mongolian Vowel Separator

Qana with Connected Final   Qana with Separated Final

The MVS has a twofold effect on shaping. On the one hand, it always selects the leftward tail form of a following letter *a* or *e*. On the other hand, it may affect the form of the preceding letter. The particular form that is taken by a letter preceding an MVS depends on the particular letter and in some cases on whether traditional or modern orthography is being used. The MVS is not needed for writing Todo, Manchu, or Sibe.

Special Shaping for Separated Suffixes. In Mongolian, Todo, Manchu, and Sibe, certain grammatical suffixes are separated from the word stem or from other suffixes by a gap. Many separated suffixes exhibit shapes that are distinct from ordinary words, and thus require special shaping.

There are many separated suffixes in Mongolian, usually occurring in masculine and feminine pairs (for example, the dative suffixes *-dur* and *-dür*), most of which require special shaping; a stem may have multiple separated suffixes. In contrast, there are only six separated suffixes for Manchu and Sibe, only one of which (*-i*) requires special shaping; stems do not have more than one separated suffix at a time.

Because separated suffixes are usually considered an integral part of the word as a whole, a line break opportunity does not normally occur before a separated suffix. The whitespace preceding the suffix is often narrower than an ordinary space, although the width may expand during justification. Prior to Unicode Version 16.0, U+202F NARROW NO-BREAK SPACE (NNBSP) was used to represent this small white-space; it retains its Script_Extensions value of "Mong" to facilitate backward compatibility. However, its role has been taken over by U+180E MONGOLIAN VOWEL SEPARATOR (MVS), which not only prevents word breaking and line breaking, but also triggers special shaping for the following separated suffix. The resulting shape depends on the particular separated suffix. Note that MVS may be preceded by another separated suffix, and may also appear between non-Mongolian characters and a separated suffix.

Normally, MVS does not provide a line breaking opportunity. However, in situations where a line is broken before a separated suffix, such as in narrow columns, it is important not to disable the special shaping triggered by MVS. This behavior may be achieved by placing the break so that the MVS character is at the start of the new line. At the beginning of the line, the MVS should affect only the shaping of the following Mongolian characters, and should display with no advance width.

Baluda. The two Mongolian *baluda* characters, U+1885 MONGOLIAN LETTER ALI GALI BALUDA and U+1886 MONGOLIAN LETTER ALI GALI THREE BALUDA, are historically related to U+0F85 TIBETAN MARK PALUTA. When used in Mongolian text rendered vertically, a *baluda* or *three baluda* character appears to the right side of the first character in a word. To simplify rendering implementations for Mongolian Ali Gali texts, the *baluda* characters have been categorized as nonspacing combining marks, rather than as letters.

Numbers. The Mongolian and Todo scripts use a set of ten digits derived from the Tibetan digits. In vertical text, numbers are traditionally written from left to right across the width of the column. In modern contexts, they are frequently rotated so that they follow the vertical flow of the text.

The Manchu and Sibe scripts do not use any special digits, although Chinese number ideographs may be employed—for example, for page numbering in traditional books.

Punctuation. Traditional punctuation marks used for Mongolian and Todo include the U+1800 MONGOLIAN BIRGA (marks the start of a passage or the recto side of a folio), U+1802 MONGOLIAN COMMA, U+1803 MONGOLIAN FULL STOP, and U+1805 MONGOLIAN FOUR DOTS (marks the end of a passage). The *birga* occurs in several different glyph forms.

In writing Mongolian and Todo, U+1806 MONGOLIAN TODO SOFT HYPHEN is used at the beginning of the second line to indicate resumption of a broken word. It func-

tions like U+2010 HYPHEN, except that U+1806 appears at the beginning of a line rather than at the end.

The Manchu script normally uses only two punctuation marks: U+1808 MONGOLIAN MANCHU COMMA and U+1809 MONGOLIAN MANCHU FULL STOP.

In modern contexts, Mongolian, Todo, and Sibe may use a variety of Western punctuation marks, such as parentheses, quotation marks, question marks, and exclamation marks. U+2048 QUESTION EXCLAMATION MARK and U+2049 EXCLAMATION QUESTION MARK are used for side-by-side display of a question mark and an exclamation mark together in vertical text. Todo and Sibe may additionally use punctuation marks borrowed from Chinese, such as U+3001 IDEOGRAPHIC COMMA, U+3002 IDEOGRAPHIC FULL STOP, U+300A LEFT DOUBLE ANGLE BRACKET, and U+300B RIGHT DOUBLE ANGLE BRACKET.

Nirugu. U+180A MONGOLIAN NIRUGU acts as a stem extender. In traditional Mongolian typography, it is used to physically extend the stem joining letters, so as to increase the separation between all letters in a word. This stretching behavior should preferably be carried out in the font rather than by the user manually inserting U+180A.

The *nirugu* may also be used to separate two parts of a compound word. For example, *altan-agula* "The Golden Mountains" may be written with the words *altan,* "golden," and *agula*, "mountains," joined together using the *nirugu*. In this usage the *nirugu* is similar to the use of hyphen in Latin scripts, but it is nonbreaking.

Syllable Boundary Marker. U+1807 MONGOLIAN SIBE SYLLABLE BOUNDARY MARKER is used to disambiguate syllable boundaries within a word. It is mainly used for writing Sibe, but may also occur in Manchu texts. In native Manchu or Sibe words, syllable boundaries are never ambiguous; when transcribing Chinese proper names in the Manchu or Sibe script, however, the syllable boundary may be ambiguous. In such cases, U+1807 may be inserted into the character sequence at the syllable boundary.

13.5.2 Mongolian Supplement: U+11660–U+1167F

The Mongolian Supplement block contains a supplemental collection of *birga* head mark signs of various shapes and orientations. These complement the basic U+1800 MONGOLIAN BIRGA.

13.6 Limbu

13.6.1 Limbu: U+1900–U+194F

The Limbu script is a Brahmic script primarily used to write the Limbu language. Limbu is a Tibeto-Burman language of the East Himalayish group and is spoken by about 200,000 persons mainly in eastern Nepal, but also in the neighboring Indian states of Sikkim and West Bengal (Darjeeling district). Its close relatives are the languages of the East Himalayish or "Kiranti" group in Eastern Nepal. Limbu is distantly related to the Lepcha (Róng) language of Sikkim and to Tibetan. Limbu was recognized as an official language in Sikkim in 1981.

The Nepali name *Limbu* is of uncertain origin. In Limbu, the Limbu call themselves *yakthuŋ*. Individual Limbus often take the surname "Subba," a Nepali term of Arabic origin meaning "headman." The Limbu script is often called "Sirijanga" after the Limbu culture-hero Sirijanga, who is credited with its invention. It is also sometimes called Kirat, *kirāta* being a Sanskrit term probably referring to some variety of non-Aryan hill-dwellers.

The oldest known writings in the Limbu script, most of which are held in the India Office Library, London, were collected in Darjeeling district in the 1850s. The modern script was developed beginning in 1925 in Kalimpong (Darjeeling district) in an effort to revive writing in Limbu, which had fallen into disuse. The encoding in the Unicode Standard supports the three versions of the Limbu script: the nineteenth-century script, found in manuscript documents; the early modern script, used in a few, mainly mimeographed, publications between 1928 and the 1970s; and the current script, used in Nepal and India (especially Sikkim) since the 1970s. There are significant differences, particularly between some of the glyphs required for the nineteenth-century and modern scripts.

Virtually all Limbu speakers are bilingual in Nepali, and far more Limbus are literate in Nepali than in Limbu. For this reason, many Limbu publications contain material both in Nepali and in Limbu, and in some cases Limbu appears in both the Limbu script and the Devanagari script. In some publications, literary coinages are glossed in Nepali or in English.

Consonants. Consonant letters and clusters represent syllable initial consonants and clusters followed by the inherent vowel, short open o ([ɔ]). Subjoined consonant letters are joined to the bottom of the consonant letters, extending to the right to indicate "medials" in syllable-initial consonant clusters. There are very few of these clusters in native Limbu words. The script provides for subjoined ◌ᤛ -ya, ◌ᤙ -ra, and ◌ᤗ -wa. Small letters are used to indicate syllable-final consonants. (See the following information on vowel length for further details.) The small letter consonants are found in the range U+1930..U+1938, corresponding to the syllable finals of native

Limbu words. These letters are independent forms that, unlike the conjoined or half-letter forms of Indian scripts, may appear alone as word-final consonants (where Indian scripts use full consonant letters and a virama). The syllable finals are pronounced without a following vowel.

Limbu is a language with a well-defined syllable structure, in which syllable-initial stops are pronounced differently from finals. Syllable initials may be voiced following a vowel, whereas finals are never voiced but are pronounced unreleased with a simultaneous glottal closure, and geminated before a vowel. Therefore, the Limbu block encodes an explicit set of ten syllable-final consonants. These are called LIMBU SMALL LETTER KA, and so on.

Vowels. The Limbu vowel system has seven phonologically distinct timbres: [i, e, ε, a, ɔ, o, u]. The vowel [ɔ] functions as the inherent vowel in the modern Limbu script. To indicate a syllable with a vowel other than the inherent vowel, a *vowel sign* is added over, under, or to the right of the initial consonant letter or cluster. Although the vowel [ɔ] is the inherent vowel, the Limbu script has a combining vowel sign ˘ that may optionally be used to represent it. Many writers avoid using this sign because they consider it redundant.

Syllable-initial vowels are represented by a vowel-carrier character, U+1900 ꄰ LIMBU VOWEL-CARRIER LETTER, together with the appropriate vowel sign. Used without a following vowel sound, the vowel-carrier letter represents syllable-initial [ɔ], the inherent vowel. The initial consonant letters have been named *ka*, *kha*, and so on, in this encoding, although they are in fact pronounced Ƨ [kɔ], ɑ [kʰɔ], and so on, and do not represent the Limbu syllables Ƨ̄ [ka], ɑ̄ [kʰa], and so on. This is in keeping with the practice of educated Limbus in writing the letter-names in Devanagari. It would have been confusing to call the vowel-carrier letter A, however, so an artificial name is used in the Unicode Standard. The native name is ꄰꢸ [ɔm].

Vowel Length. Vowel length is phonologically distinctive in many contexts. Length in open syllables is indicated by writing U+193A ˙˙ LIMBU SIGN KEMPHRENG, which looks like the diaeresis sign, over the initial consonant or cluster: Ӟ̈ *tā*.

In closed syllables, two different methods are used to indicate vowel length. In the first method, vowel length is not indicated by *kemphreng*. The syllable-final consonant is written as a full form (that is, like a syllable-initial consonant), marked by U+193B ˳ LIMBU SIGN SA-I: ꄰ̰Ƶ *pān* "speech." This sign marks vowel length in addition to functioning as a virama by suppressing the inherent vowel of the syllable-final consonant. This method is widely used in Sikkim.

In the second method, which is in use in Nepal, vowel length is indicated by *kemphreng*, as for open syllables, and the syllable-final consonant appears in "small" form without *sa-i*: ꄰ̈ꢼ *pān* "speech." Writers who consistently follow this practice reserve the use of *sa-i* for syllable-final consonants that do not have small forms,

regardless of the length of the syllable vowel: ইগ্নী *nesse* "it lay," ঀিঢ় *lāb* "moon." Because almost all of the syllable finals that normally occur in native Limbu words have small forms, *sa-i* is used only for consonant combinations in loan words and for some indications of rapid speech.

U+193B ◌ LIMBU SIGN SA-I is based on the Indic virama, but for a majority of current writers it has a different semantics because it indicates the length of the preceding vowel in addition to "killing" the inherent vowel of consonants functioning as syllable finals. It is therefore not suitable for use as a general virama as used in other Brahmic scripts in the Unicode Standard.

Glottalization. U+1939 LIMBU SIGN MUKPHRENG represents glottalization. *Mukphreng* never appears as a syllable initial. Although some linguists consider that word-final nasal consonants may be glottalized, this is never indicated in the script; *mukphreng* is not currently written after final consonants. No other syllable-final consonant clusters occur in Limbu.

Collating Order. There is no universally accepted alphabetical order for Limbu script. One ordering is based on the Limbu dictionary edited by Bairagi Kainla, with the addition of the obsolete letters, whose positions are not problematic. In Sikkim, a somewhat different order is used: the letter ই *na* is placed before ৩ *ta*, and the letter ঘ *gha* is placed at the end of the alphabet.

Glyph Placement. The glyph positions for Limbu combining characters are summarized in *Table 13-8*.

Table 13-8. Positions of Limbu Combining Characters

Syllable	Glyphs	Code Point Sequence
ta	৩	190B 1920
ti	৩	190B 1921
tu	৩	190B 1922
tee	৩৯	190B 1923
tai	৩৯৯	190B 1924
too	৩৯	190B 1925
tau	৩৯৯	190B 1926
te	৩	190B 1927
to	৩	190B 1928
tya	৩ᵥ	190B 1929
tra	৩	190B 192A
twa	৩ᵃ	190B 192B
tak	৩ᶜ	190B 1930
taŋ	৩₀	190B 1931

taṁ	३ँ	190B 1932
tat	३ॢ	190B 1933
tan	३ᵕ	190B 1934
tap	३५	190B 1935
tam	३ॶ	190B 1936
tar	३ॢ	190B 1937
tal	३ᵘ	190B 1938
tā	३ँ	190B 1920 193A
tī	३ँ	190B 1921 193A

Punctuation. The main punctuation mark used is the double vertical line, U+0965 DEVANAGARI DOUBLE DANDA. U+1945 ᥅ LIMBU QUESTION MARK and U+1944 ᥄ LIMBU EXCLAMATION MARK have shapes peculiar to Limbu, especially in Sikkimese typography. They are encoded in the Unicode Standard to facilitate the use of both Limbu and Devanagari scripts in the same documents. U+1940 ᥀ LIMBU SIGN LOO is used for the exclamatory particle *lo*. This particle is also often simply spelled out ᤗᤥ.

Digits. Limbu digits have distinctive forms and are assigned code points because Limbu and Devanagari (or Limbu and Arabic-Indic) numbers are often used in the same document.

13.7 Meetei Mayek

13.7.1 Meetei Mayek: U+ABC0–U+ABFF

Meetei Mayek is a script used for Meetei, a Tibeto-Burman language spoken primarily in Manipur, India. The script originates from the Tibetan group of scripts, which in turn derive from Gupta Brahmi. The script has experienced a recent resurgence in use. The modern-day Meetei Mayek script is made up of a core repertoire of 27 letters, alongside letters and symbols for final consonants, dependent vowel signs, punctuation, and digits.

The name "Meetei Mayek" is used in official documentation in Manipur. The script may also appear with other spellings and names, such as "Meitei Mayek," "Methei," "Meetei," or the older "Manipuri."

Structure. Meetei Mayek is a Brahmic script with consonants bearing the inherent vowel and vowel matras modifying it. However, unlike most other Brahmi-derived scripts, Meetei Mayek employs explicit final consonants which contain no final vowels.

Meetei Mayek has a killer character, U+ABED MEETEI MAYEK APUN IYEK, which may be used to indicate the lack of an inherent vowel when no explicit consonant letter exists. In modern orthography, the killer does not cause conjunct formation and is always visible. The use of the killer is optional in spelling; for example, while ꯀꯔ may be read *kara* or *kra*, ꯀ꯭ꯔ must be read *kra*. In the medial position, the glyph of the killer usually extends below the killed letter and the following letter.

Vowel Letters. In modern use, only three vowel characters, U+ABD1 MEETEI MAYEK LETTER ATIYA, U+ABCF MEETEI MAYEK LETTER I, and U+ABCE MEETEI MAYEK LETTER UN (= *u*), may appear initially or word-internally. Other vowels without independent forms are represented by vowel matras applied to U+ABD1 MEETEI MAYEK LETTER ATIYA. In modern orthography, the seven dependent vowel signs and the *anusvara,* U+ABEA MEETEI MAYEK VOWEL SIGN NUNG, located from U+ABE3..U+ABEA, are used with consonants.

Syllable initial combinations for vowels can occur in modern usage to represent diphthongs.

Final Consonants. There are three ways to indicate final consonants in Meetei Mayek: by the eight explicit final consonant letters, by U+ABEA MEETEI MAYEK VOWEL SIGN NUNG, which acts as an *anusvara*, or by U+ABCE MEETEI MAYEK LETTER UN, which may act as a final consonant without modification.

Abbreviations. Unusual abbreviations composed of a single consonant and more than one matra may occur in a manner similar that found in Tibetan. In such cases, the vowel matra may occur at the end of a word.

Order. The order of the first 18 Meetei letters is based upon the parts of the body. This system is discussed in a religious manuscript, the *Wakoklon hilel thilel salai amailon pukok puya* (commonly referred to as the *Wakoklon puya*), which describes the letters, and relates them to the corresponding body part. The Meetei Mayek letter *kok*, for example, means "head," *sam* designates "hair-parting," and *lai* is "forehead." The last 9 letters,*gok*, *jham*, *rai*, and so forth, derive from a subset of the original 18. The ordering system employed today differs from the Brahmi-based order, which relies on the point of articulation.

Punctuation. The modern Meetei Mayek script uses two punctuation marks in addition to the killer. U+ABEB MEETEI MAYEK CHEIKHEI functions as a double danda mark. U+ABEC MEETEI MAYEK LUM IYEK is a heavy tone mark, used to orthographically distinguish words which would otherwise not be differentiated.

Digits. Meetei Mayek has a unique set of ten digits for zero to nine encoded in the range at U+ABF0..U+ABF9.

13.7.2 Meetei Mayek Extensions: U+AAE0–U+AAFF

The Meetei Mayek Extensions block contains additional characters needed to represent the historical orthographies of Meetei. The block includes nine consonants, encoded in the range U+AAE2..U+AAEA, two independent vowel signs (U+AAE0 MEETEI MAYEK LETTER E and U+AAE1 MEETEI MAYEK LETTER O), and five dependent vowels signs in the range U+AAEB..U+AAEF.

U+AAF6 MEETEI MAYEK VIRAMA should be used to represent conjuncts that may occur in historical texts. The *virama* is not visibly rendered, but it behaves as in other Brahmi-derived scripts. For example, the conjunct /ṅha/ is represented by the sequence <ABC9, AAF6, ABCD>.

This block also includes two punctuation marks, U+AAF0 MEETEI MAYEK CHEIKHAN and U+AAF1 MEETEI MAYEK AHANG KHUDAM. The *cheikhan* is a single *danda*, and *ahang khudam* is a question mark. U+AAF2 MEETEI MAYEK ANJI is a philosophical sign indicating auspiciousness. Finally, two repetition marks are included in the block: U+AAF3 MEETEI MAYEK SYLLABLE REPETITION MARK and U+AAF4 MEETEI MAYEK WORD REPETITION MARK.

13.8 Mro

13.8.1 Mro: U+16A40–U+16A6F

The Mro script was invented in the 1980s. It is used to write the Mro (or Mru) language, a language of the Mruic branch of Tibeto-Burman spoken in Southeastern Bangladesh and neighboring areas of Myanmar. (This language is distinct from Mro-Khimi, a language of the Kukish branch of Tibeto-Burman spoken in Myanmar.)

The Mro script is unrelated to any other script. Some of the letters of the Mro alphabet have a visual similarity to letters from other alphabets, but such similarities are coincidental.

Structure. The Mro script is a left-to-right alphabet with no combining characters or tone marks. Some sounds are represented by more than one letter.

Character Names. Consonant letter names are traditional, based on phonetic transcription.

Digits. Mro has a script-specific set of digits.

Punctuation. There are two script-specific punctuation characters, U+16A6E MRO DANDA and U+16A6F MRO DOUBLE DANDA. Words are separated by spaces.

Two of the Mro letters are used as abbreviations. U+16A5E MRO LETTER TEK can be used instead of the word "tek," meaning "quote." U+16A5C MRO LETTER HAI can be used for various groups of letters.

13.9 Warang Citi

13.9.1 Warang Citi: U+118A0–U+118FF

The Warang Citi script is used to write the Ho language. Ho is a North Munda language. Warang Citi was devised by community leader Lako Bodra as an improvement over scripts used by Christian missionary linguists. Speakers of Ho live in the Indian states of Odisha (formerly Orissa) and Jharkhand. There are at present two publications in the script: a yearly magazine and a biweekly publication.

The Ho community is primarily an oral community, with an emergent literary tradition. Many Ho speakers do not write their language in any form. In some areas, Ho speakers use the Devanagari script or Warang Citi, in other locations they use the Oriya (now officially known as Odia) script or Warang Citi. There are also people who use Latin letters to write Ho on an ad-hoc basis.

Structure. Warang Citi is an alphabet, written from left to right. Unlike many other Indic scripts, vowels are written as full letters, with no vowel-modifiers. However, consonants may have an inherent vowel; it typically is pronounced [a] or [ɔ], and less often [ɛ], but this vowel does not occur in final position in a word. Because these inherent vowels are not written explicitly, there can be ambiguity in the reading of certain words.

Warang Citi has no regular system of conjuncts nor an explicit virama. However, a small number of conjunct forms are used; most of these represent doubled consonants. The choice of a conjunct form does not appear to be predictable. The recommended mechanism for representing these conjuncts is to make use of U+200D ZERO WIDTH JOINER.

Warang Citi uses case distinctions, so both uppercase and lowercase letters are encoded.

The script does not include a diacritical mark for *anusvara* as in Devanagari, but rather has a separate character, U+118C0 WARANG CITI SMALL LETTER NGAA.

Digits and Numbers. Warang Citi has a set of digits and numbers, but the orthographic conventions for writing numbers have not yet stabilized. European digits are also used, though not consistently.

Punctuation. Warang Citi uses Latin punctuation. There is no script-specific punctuation.

13.10 Ol Chiki

13.10.1 Ol Chiki: U+1C50–U+1C7F

The Ol Chiki script was invented by Pandit Raghunath Murmu in the first half of the 20th century CE to write Santali, a Munda language of India. The script is also called Ol Cemet', Ol Ciki, or simply Ol. Santali has also been written with the Devanagari, Bengali, and Oriya scripts, as well as the Latin alphabet.

Various dialects of Santali are spoken by 5.8 million people, with 25% to 50% literacy rates, mostly in India, with a few in Nepal or Bangladesh. The Ol Chiki script is used primarily for the southern dialect of Santali as spoken in the Odishan Mayurbhañj district. The script has received some official recognition by the Odishan government.

Ol Chiki has recently been promoted by some Santal organizations, with uncertain success, for use in writing certain other Munda languages in the Chota Nagpur area, as well as for the Dravidian Dhangar-Kudux language.

Structure. Ol Chiki is an alphabet written from left to right, and has none of the structural properties of the *abugidas* typical for other Indic scripts. Vowels are written with standalone letters. A number of modifier letters are used to indicate tone, nasalization, vowel length, and deglottalization. There are no combining characters in the script. (See *Section 13.12, Nag Mundari*, for a similarly structured script.)

Modifier Letters. The southern dialect of Santali has only six vowels, each represented by a single vowel letter. The Santal Parganas dialect, on the other hand, has eight or nine vowels. The extra vowels for Santal Parganas are represented by a sequence of one of the vowel letters U+1C5A, U+1C5F, or U+1C6E followed by the diacritic modifier letter, U+1C79 OL CHIKI GAAHLAA TTUDDAAG, displayed as a baseline dot.

Nasalization is indicated by the modifier letter, U+1C78 OL CHIKI MU TTUDDAG, displayed as a raised dot. This mark can follow any vowel, long or short.

When the vowel diacritic and nasalization occur together, the combination is represented by a separate modifier letter, U+1C7A OL CHIKI MU-GAAHLAA TTUDDAAG, displayed as both a baseline and a raised dot. The combination is treated as a separate character and is entered using a separate key on Ol Chiki keyboards.

U+1C7B OL CHIKI RELAA is a length mark, which can be used with any oral or nasalized vowel.

Glottalization. U+1C7D OL CHIKI AHAD is a special letter indicating the deglottalization of an Ol Chiki consonant in final position. This unique feature of the writing system preserves the morphophonemic relationship between the glottalized (ejective)

and voiced equivalents of consonants. For example, U+1C5C OL CHIKI LETTER AG represents an ejective [k'] when written in word-final position, but voiced [g] when written word-initially. A voiced [g] in word-final position is written with the deglottalization mark as a sequence: <U+1C5C OL CHIKI LETTER AG, U+1C7D OL CHIKI AHAD>.

U+1C7C OL CHIKI PHAARKAA serves the opposite function. It is a "glottal protector." When it follows one of the four ejective consonants, it preserves the ejective sound, even in word-initial position followed by a vowel.

Aspiration. Aspirated consonants are written as digraphs, with U+1C77 OL CHIKI LETTER OH as the second element, indicating the aspiration.

Ligatures. Ligatures are not a normal feature of printed Ol Chiki. However, in handwriting and script fonts, letters form cursive ligatures with the deglottalization mark, U+1C7D OL CHIKI AHAD.

Punctuation. Western-style punctuation, such as the comma, exclamation mark, question mark, and quotation marks are used in Ol Chiki text. U+002E " . " FULL STOP is not used, because it is visually confusable with the modifier letter U+1C79 OL CHIKI GAAHLAA TTUDDAAG.

The *danda*, U+1C7E OL CHIKI PUNCTUATION MUCAAD, is used as a text delimiter in prose. The *danda* and the *double danda*, U+1C7F OL CHIKI PUNCTUATION DOUBLE MUCAAD, are both used in poetic text.

Digits. The Ol Chiki script has its own set of digits, encoded in the Ol Chiki block in the range U+1C50..U+1C59.

13.11 Ol Onal

13.11.1 Ol Onal: U+1E5D0–U+1E5FF

The Ol Onal script was invented by Mahendra Nath Sardar toward the end of the 20th century CE to write Bhumij, a Munda language of India. The script is also known as Bhumij Lipi or Bhumij Onal. Bhumij is also written with the Devanagari, Bengali, and Oriya scripts, as well as with the Latin script.

The Bhumij language is an Austroasiatic language of the North Munda group. Mundari and Bhumij are closely related and mutually intelligible, but with considerable differences.

Ol Onal was originally designed as a bicameral alphabetical script: the uppercase letters are called Ol Onal, while the lowercase letters are known as Galang Onal. However, all the teaching and printed materials in Bhumij have been produced using Ol Onal (that is, capital letters). There is no record of Galang Onal ever being used, so lowercase letters are not encoded.

Structure. Ol Onal is an alphabet written from left to right, and has none of the structural properties of the *abugidas* typical for other Indic scripts. Vowels are written with standalone letters. There are three additional signs, *mu*, *ikir*, and *hoddond*, collectively referred to as *ṭiḍaḥ*. (See *Section 13.10, Ol Chiki*, for a similarly structured script.)

Signs. Nasalization is indicated by U+1E5EE ○ OL ONAL SIGN MU. It appears as a dot positioned above vowels. The nonspacing mark U+1E5EF ○ OL ONAL SIGN IKIR indicates a long phoneme, and currently it is only used below the letter U+1E5D6 θ‖ OL ONAL LETTER A. The letter a can carry both nasalization and lengthening signs simultaneously.

U+1E5F0 ৪ OL ONAL SIGN HODDOND is a letter indicating glottalization. It can follow the letters U+1E5D8 @ OL ONAL LETTER AB and U+1E5E5 ⟨ᴡ⟩ OL ONAL LETTER UJ.

Punctuation. Ol Onal uses U+0964 । DEVANAGARI DANDA to end sentences. Western-style punctuation is also used, such as comma, question mark, dash, colon, parentheses, slash, and backslash, as well as the percent sign.

A separate U+1E5FF ° OL ONAL ABBREVIATION SIGN is attested for use to indicate an abbreviation.

Digits. The Ol Onal script has its own set of digits, encoded in the range U+1E5F1..U+1E5FA.

13.12 Nag Mundari

13.12.1 Nag Mundari: U+1E4D0–U+1E4FF

Nag Mundari is a script used to write the Mundari language, spoken primarily in the states of Jharkhand, West Bengal and Odisha in India. The block is named after the script's creator Rohidas Singh Nag, while in the community the script is most commonly known as "Mundari Bani." Less common names for the script include "Mundari Lipi" and "Hoḍo Jagar." While the earliest work on the script began in the early 1950s, this block encodes a reformed version of the script from 2008 by Bharat Munda Samaj and Mundari Samaj Sanwar Jamda.

Structure. Nag Mundari is an alphabetic script written left to right, with no attested ligatures. Nag Mundari is alphabetic and has none of the structural properties of the abugidas typical for other Indic scripts. There are separate letters representing consonants and vowels. (See *Section 13.10, Ol Chiki*, for a similarly structured script.)

Signs. A collection of extended marks in the range U+1E4EB..U+1E4EF, referred to as *tong*, are used to indicate nasalization, vowel length, phonetic changes and sounds that are not native to the writing system.

Vowel nasalization is indicated by the sign U+1E4EC ◌ NAG MUNDARI SIGN MUHOR, written to the top right of the vowel with which it combines.

Long vowels are indicated by the sign U+1E4ED ◌ NAG MUNDARI SIGN TOYOR, although the lack of *toyor* does not necessarily imply a short vowel. It is written to the top right of the vowel.

U+1E4EE ◌ NAG MUNDARI SIGN IKIR is used to denote that a vowel is preceded by the /w/ sound.

U+1E4EF ◌ NAG MUNDARI SIGN SUTUH is used for producing characters to represent sounds not native to the Mundari language. The sign may be used with any Nag Mundari letter, including vowels and consonants.

The modifier letter U+1E4EB ꓘ NAG MUNDARI SIGN OJOD can be used variously to denote checked consonants, or to indicate consonant gemination when transcribing other languages. Note that *ojod* should be distinguished from the visually similar U+1E4E1 ꓘ NAG MUNDARI LETTER UD.

Punctuation. Western-style punctuation, such as the full stop, comma, exclamation mark, question mark, quotation marks and hyphen-minus are used in Nag Mundari text. Hyphen or hyphen-minus can be used to represent the dash known as *eced* in Nag Mundari, which marks compound words, reduplications and similar constructions.

Digits. The Nag Mundari script has its own set of digits encoded in the Nag Mundari block in the range U+1E4F0..U+1E4F9. European, Devanagari, Bengali or Oriya digits are also used.

13.13 Chakma

13.13.1 Chakma: U+11100–U+1114F

The Chakma people, who live in southeast Bangladesh near Chittagong City, as well as in parts of India such as Mizoram, Assam, Tripura, and Arunachal Pradesh, speak an Indo-European language also called Chakma. The language, spoken by about 500,000 people, is related to the Assamese, Bengali, Chittagonian, and Sylheti languages.

The Chakma script is Brahmi-derived, and is sometimes also called *Ajhā pāṭh* or *Ojhopath*. There are some efforts to adapt the Chakma script to write the closely related Tanchangya language. One of the interesting features of Chakma writing is that *candrabindu* (*cānaphudā*) can be used together with *anusvara* (*ekaphudā*) and *visarga* (*dviphudā*).

Independent Vowels. Like other Brahmi-derived scripts, Chakma uses consonant letters that contain an inherent vowel. Consonant clusters are written with conjunct characters, while a visible "vowel killer" (called the *maayyaa*) shows the deletion of the inherent vowel when there is no conjunct. There are four independent vowels: U+11103 CHAKMA LETTER AA /ā/, U+11104 CHAKMA LETTER I /i/, U+11105 CHAKMA LETTER U /u/, and U+11106 CHAKMA LETTER E /e/. Other vowels in the initial position are formed by adding a dependent vowel sign to the independent vowel /ā/, to form vowels such as /ī/, /ō/, /ai/, and /oi/.

Vowel Killer and Virama. Like the Myanmar script and the characters used to write historic Meetei Mayek, Chakma is encoded with two vowel-killing characters to conform to modern user expectations. In most cases vowels are killed with the use of U+11134 CHAKMA MAAYYAA. In addition to that visible killer, U+11133 CHAKMA VIRAMA joins two consonant letters to form a conjunct, which may be either a subjoined stack or a ligature. Whether a conjunct is required or not is part of the spelling of a word.

U+11134 CHAKMA MAAYYAA is also used to indicate geminated consonants, in which case the consonant letter is typically followed by two combining marks, first the *maayyaa* and then a vowel sign.

In 2001, an orthographic reform was recommended in the book *Cāṅmā pattham pāt*, limiting the standard repertoire of conjuncts to those composed with the five letters U+11121 CHAKMA LETTER YAA /yā/, U+11122 CHAKMA LETTER RAA /rā/, U+11123 CHAKMA LETTER LAA /lā/, U+11124 CHAKMA LETTER WAA /wā/, and U+1111A CHAKMA LETTER NAA /nā/.

Chakma Fonts. Chakma fonts by default should display the subjoined form of letters that follow virama to ensure legibility.

Punctuation. Chakma has a single and double danda. There is also a unique question mark and a section mark, *phulacihna*.

Digits. A distinct set of digits is encoded for Chakma. Bengali digits are also used with Chakma. Myanmar digits are used with the Chakma script when writing Tanchangya.

13.14 Lepcha

13.14.1 Lepcha: U+1C00–U+1C4F

Lepcha is a Sino-Tibetan language spoken by people in Sikkim and in the West Bengal state of India, especially in the Darjeeling district, which borders Sikkim. The Lepcha script is a writing system thought to have been invented around 1720 CE by the Sikkim king Phyag-rdor rNam-rgyal ("Chakdor Namgyal," born 1686). Both the language and the script are also commonly known by the term *Rong*.

Structure. The Lepcha script was based directly on the Tibetan script. The letter-forms are obviously related to corresponding Tibetan letters. However, the *dbu-med* Tibetan precursors to Lepcha were originally written in vertical columns, possibly influenced by Chinese conventions. When Lepcha was invented it changed the *dbu-med* text to a left-to-right, horizontal orientation. In the process, the entire script was effectively rotated ninety degrees counterclockwise, so that the letters resemble Tibetan letters turned on their sides. This reorientation resulted in some letters which are nonspacing marks in Tibetan becoming spacing letters in Lepcha. Lepcha also introduced its own innovations, such as the use of diacritical marks to represent final consonants.

The Lepcha script is an *abugida*: the consonant letters have an inherent vowel, and dependent vowels (*matras*) are used to modify the inherent vowel of the consonant. No virama (or vowel killer) is used to remove the inherent vowel. Instead, the script has a separate set of explicit final consonants which are used to represent a consonant with no inherent vowel.

Vowels. Initial vowels are represented by the neutral letter U+1C23 LEPCHA LETTER A, followed by the appropriate dependent vowel. U+1C23 LEPCHA LETTER A thus functions as a vowel carrier.

The dependent vowel signs in Lepcha always follow the base consonant in logical order. However, in rendering, three of these dependent vowel signs, *-i*, *-o*, and *-oo*, reorder to the left side of their base consonant. One of the dependent vowel signs, *-e*, is a nonspacing mark which renders below its base consonant.

Medials. There are three medial consonants, or glides: *-ya*, *-ra*, and *-la*. The first two are represented by separate characters, U+1C24 LEPCHA SUBJOINED LETTER YA and U+1C25 LEPCHA SUBJOINED LETTER RA. These are called "subjoined", by analogy with the corresponding letters in Tibetan, which actually do join below a Tibetan consonant, but in Lepcha these are spacing forms which occur to the right of a consonant letter and then ligate with it. These two medials can also occur in sequence to form a composite medial, *-rya*. In that case both medials ligate with the preceding consonant.

On the other hand, Lepcha does not have a separate character to represent the medial *-la*. Phonological consonant clusters of the form *kla*, *gla*, *pla*, and so on simply have separate, atomic characters encoded for them. With few exceptions, these letters for phonological clusters with the medial *-la* are independent letterforms, not clearly related to the corresponding consonants without *-la*.

Retroflex Consonants. The Lepcha language contains three retroflex consonants: [ṭ], [ṭh], and [ḍ]. Traditionally, these retroflex consonants have been written in the Lepcha script with the syllables *kra*, *hra*, and *gra*, respectively. In other words, the *retroflex t* would be represented as <U+1C00 LEPCHA LETTER KA, U+1C25 LEPCHA SUBJOINED LETTER RA>. To distinguish such a sequence representing a *retroflex t* from a sequence representing the actual syllable [kra], it is common to use the *nukta* diacritic sign, U+1C37 LEPCHA SIGN NUKTA. In that case, the *retroflex t* would be visually distinct, and would be represented by the sequence <U+1C00 LEPCHA LETTER KA, U+1C37 LEPCHA SIGN NUKTA, U+1C25 LEPCHA SUBJOINED LETTER RA>. Recently, three newly invented letters have been added to the script to unambiguously represent the retroflex consonants: U+1C4D LEPCHA LETTER TTA, U+1C4E LEPCHA LETTER TTHA, and U+1C4F LEPCHA LETTER DDA.

Ordering of Syllable Components. Dependent vowels and other signs are encoded after the consonant to which they apply. The ordering of elements is shown in more detail in *Table 13-9*.

<p align="center">**Table 13-9.** Lepcha Syllabic Structure</p>

Class	Example	Encoding
consonant, letter a	€	[U+1C00..U+1C23, U+1C4D..U+1C4F]
nukta	◌	U+1C37
medial -ra	◌)	U+1C25
medial -ya	◌ν	U+1C24
dependent vowel	(◌	[U+1C26..U+1C2C]
final consonant sign	◌	[U+1C2D..U+1C35]
syllabic modifier	◌̃	U+1C36

Rendering. Most final consonants consist of nonspacing marks rendered above the base consonant of a syllable.

The combining mark U+1C36 LEPCHA SIGN RAN occurs after the inherent vowel *-a* or the dependent vowel *-i*. When it occurs together with a final consonant sign, the *ran* sign renders above the sign for that final consonant.

The two final consonants representing the velar nasal occur in complementary contexts. U+1C34 LEPCHA CONSONANT SIGN NYIN-DO is only used when there is no dependent vowel in the syllable. U+1C35 LEPCHA CONSONANT SIGN KANG is used

instead when there is a dependent vowel. These two consonant signs are rendered to the left of the base consonant. If used with a left-side dependent vowel, the glyph for the *kang* is rendered to the left of the dependent vowel. This behavior is understandable because these two marks are derived from the Tibetan analogues of the Brahmic *bindu* and *candrabindu*, which normally stand above a Brahmic *aksara*.

Digits. The Lepcha script has its own, distinctive set of digits.

Punctuation. Currently the Lepchas use traditional punctuation marks only when copying the old books. In everyday writing they use common Western punctuation marks such as comma, full stop, and question mark.

The traditional punctuation marks include a script-specific *danda* mark, U+1C3B LEPCHA PUNCTUATION TA-ROL, and a *double danda*, U+1C3C LEPCHA PUNCTUATION NYET THYOOM TA-ROL. Depending on style and hand, the Lepcha *ta-rol* may have a glyph appearance more like its Tibetan analogue, U+0F0D TIBETAN MARK SHAD.

13.15 Saurashtra

13.15.1 Saurashtra: U+A880–U+A8DF

Saurashtra is an Indo-European language, related to Gujarati and spoken by about 310,000 people in southern India. The Telugu, Tamil, Devanagari, and Saurashtra scripts have been used to publish books in Saurashtra since the end of the 19th century. At present, Saurashtra is most often written in the Tamil script, augmented with the use of superscript digits and a colon to indicate sounds not available in the Tamil script.

The Saurashtra script is of the Brahmic type. Early Saurashtra text made use of conjuncts, which can be handled with the usual Brahmic shaping rules. The modernized script, developed in the 1880s, has undergone some simplification. Modern Saurashtra does not use complex consonant clusters, but instead marks a killed vowel with a visible virama, U+A8C4 SAURASHTRA SIGN VIRAMA. An exception to the non-occurrence of complex consonant clusters is the conjunct *kṣa*, formed by the sequence <U+A892, U+A8C4, U+200D, U+A8B0>. This conjunct is sorted as a unique letter in older dictionaries. Apart from its use to form *kṣa*, the virama is always visible by default in modern Saurashtra. If necessary, U+200D ZERO WIDTH JOINER may be used to force conjunct behavior.

The Unicode encoding of the Saurashtra script supports both older and newer conventions for writing Saurashtra text.

Glyph Placement. The vowel signs (*matras*) in Saurashtra follow the consonant to which they are applied. The long and short -i vowels, however, are typographically joined to the top right corner of their consonant. Vowel signs are also applied to U+A8B4 SAURASHTRA CONSONANT SIGN HAARU.

Digits. The Saurashtra script has its own set of digits. These are separately encoded in the Saurashtra block.

Punctuation. Western-style punctuation, such as comma, full stop, and the question mark are used in modern Saurashtra text. U+A8CE SAURASHTRA DANDA is used as a text delimiter in traditional prose. U+A8CE SAURASHTRA DANDA and U+A8CF SAURASHTRA DOUBLE DANDA are used in poetic text.

Saurashtra Consonant Sign Haaru. The character U+A8B4 SAURASHTRA CONSONANT SIGN HAARU, transliterated as "H", is unique to Saurashtra, and does not have an equivalent in the Devanagari, Tamil, or Telugu scripts. It functions in some regards like the Tamil *aytam*, modifying other letters to represent sounds not found in the basic Brahmic alphabet. It is a dependent consonant and is thus classified as a consonant sign in the encoding.

13.16 Masaram Gondi

13.16.1 Masaram Gondi: U+11D00–U+11D5F

The Masaram Gondi script was created in 1918 by Munshi Mangal Singh Masaram of Madhya Pradesh, India to write the Gondi language, a Dravidian language spoken in central and southeastern India. The Gondi language, which is typically written with the Devanagari or Telugu scripts, also has another script associated with it, Gunjala Gondi, which is unrelated to the Masaram Gondi script. Masaram Gondi is actively used today in handwritten and printed materials.

Structure. Masaram Gondi is an *abugida*, written left to right. While not historically related to Brahmi, its general structure is similar to that of other Brahmi-derived Indic scripts. Masaram Gondi uses a conjunct-forming character (*virama*) to invoke conjoined consonants. In addition, there is an explicit vowel-killing character (*halanta*).

Consonants. Consonants have an inherent /a/ vowel, which is graphically represented by a horizontal stroke extending rightward from the right-hand edge of each consonant letter. A bare consonant (with no vowel) is represented by removing this stroke. Alternatively, consonants that appear in word-final position can be represented by the addition of U+11D44 MASARAM GONDI SIGN HALANTA. This use of halanta is a recent innovation employed by some modern users.

Conjuncts. Consonant clusters in Masaram Gondi are represented by the sequence <C, *virama*, C>. The consonants in a cluster appear in a linear sequence as bare forms without the horizontal stroke, except the last consonant, which retains its original form, as shown in *Figure 13-9*.

Figure 13-9. Masaram Gondi Consonant Clusters

O- *ka*	→	O- *ka*
O- *ka* + 🔲 *virama* + ∪- *ta*	→	O∪- *kta*
O- *ka* + 🔲 *virama* + ∪- *ta* + 🔲 *virama* + 8- *va*	→	O∪8- *ktva*

There are a few exceptions to the consonant cluster formation rule: the conjuncts *kssa*, *jyna*, and *tra* are atomically encoded, whereas consonant clusters with U+11D26 MASARAM GONDI LETTER RA have special contextual forms. When *ra* occurs as the first consonant in a cluster and does not mark a morphological boundary, it is generally rendered with U+11D46 MASARAM GONDI REPHA. *Repha* is represented in logical order at the beginning of a cluster, and does not interact with any combining signs. When *ra* appears first in a cluster and marks a morphological distinction, the bare consonant appears. There is also a cluster-final form of *ra*, a combining sign

called U+11D47 *ra-kara*. The *ra-kara* appears in logical order before any vowel sign. Neither *repha* nor *ra-kara* interact with the *virama*. Details are shown in *Figure 13-10*.

Figure 13-10. Rendering of *ra* in Masaram Gondi

∩ *repha* + ੦– *ka*	→	੦ᵃ*rka*
੦– *ka* + ○ı *ra-kara*	→	੦ᵣ*kra*
◊– *ra* + ○̣ *virama* + ੦– *ka*	→	◊◊੦– *rka*
੦– *ka* + ○̣ *virama* + ◊– *ra*	→	੦◊◊– *kra*

Various Signs. Masaram Gondi uses various signs, as summarized in *Table 13-10*.

Table 13-10. Various Signs in Masaram Gondi

Character	Use
11D40 MASARAM GONDI SIGN ANUSVARA	Indicates nasalization
11D41 MASARAM GONDI SIGN VISARGA	Used to represent Sanskrit words
11D42 MASARAM GONDI SIGN NUKTA	Used to indicate sounds for which distinct characters in Masaram Gondi do not exist, such as loanwords
11D43 MASARAM GONDI SIGN CANDRA	Used to transcribe vowel sounds not native to Gondi

Digits and Punctuation. Masaram Gondi has a full set of decimal digits. There are no script-specific marks of punctuation. For dandas, Masaram Gondi uses U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA.

13.17 Gunjala Gondi

13.17.1 Gunjala Gondi: U+11D60–U+11DAF

Gunjala Gondi is named after the village in the Adilabad district of the southern Indian state of Telegana, where manuscripts in the script were found. The script, also called Koytura Gunjala Lipi, is used to write the Adilabad dialect of the Gondi language, which is a member of the Dravidian family. Gunjala Gondi is taught today and appears in publications, but the dialect is more commonly written in Telugu or Devanagari. Although the Gunjala Gondi script is formally unrelated to Masaram Gondi and other scripts, it strongly resembles the Modi script in appearance and structure.

Structure. Gunjala Gondi is an *abugida*, structurally based on the Brahmi model. The vowel letters follow the order found in Brahmi-derived scripts, but the consonants appear in a different order: the first consonant is *ya*, instead of *ka*, for example. The script uses a *virama* to create conjuncts, but it does not suppress the inherent vowel. The script is written from left to right.

Consonants. The script does not have letters for palatal *na* and retroflex *na*. U+11D7A GUNJALA GONDI LETTER NA and U+11D95 GUNJALA GONDI SIGN ANUSVARA are used to represent these letters. The script does not have distinct letters for palatal *sa* and retroflex *sa*, so U+11D89 GUNJALA GONDI LETTER SA is used to represent those letters. No special forms of *ra* appear in Gunjala Gondi. The half-form of U+11D88 GUNJALA GONDI LETTER RA is used in the cluster-initial position.

Vowels. Unlike scripts used to write other Dravidian languages, Gunjala Gondi does not distinguish between long and short /e/ or /o/. U+11D67 GUNJALA GONDI LETTER EE designates both the short and the long /e/, and U+11D6A GUNJALA GONDI LETTER OO is used for short and long /o/. For the dependent vowel signs, the script similarly does not distinguish the long and short vowels /e/ and /o/, instead using U+11D90 GUNJALA GONDI VOWEL SIGN EE and U+11D93 GUNJALA GONDI VOWEL SIGN OO, respectively.

Consonant Conjuncts. Consonant clusters are represented with a *virama* between the consonants in the conjunct. The initial and medial consonants in a cluster are written horizontally with half-forms, but the final consonant appears in its full form.

Figure 13-11. Gunjala Gondi Conjunct Formation

Conjuncts composed of a consonant and the vowels signs *-aa*, *-oo*, and *-au* are usually written as ligatures with a modified form of the consonant.

Digits and Punctuation. Gunjala Gondi has a full set of decimal digits in the range U+11DA0..U+11DA9. Gunjala Gondi uses *dandas* and European punctuation, such as middle dots, periods, and colons to mark word and sentence boundaries. Gunjala Gondi uses U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA.

Other Signs. U+11D95 GUNJALA GONDI SIGN ANUSVARA indicates nasalization. U+11D96 GUNJALA GONDI SIGN VISARGA represents post-vocalic aspiration in words of Sanskrit origin. The *om* sign is encoded at U+11D98.

13.18 Wancho

13.18.1 Wancho: U+1E2C0–U+1E2FF

The Wancho script was devised between 2001 and 2012 by Banwang Losu, a teacher at a government middle school in his home village in Arunachal Pradesh, India; it is taught today in schools. The Wancho language is a Sino-Tibetan language that has some 50,000 speakers and is used chiefly in the southeast of Arunachal Pradesh, as well as in Assam and Nagaland, and in the countries of Myanmar and Bhutan.

Structure. Wancho is a simple left-to-right alphabetic script comprised of letters which represent both consonants and vowels. Diacritical marks are used on vowel letters to indicate tone.

Tones. There are four tone marks in Wancho:

- U+1E2EC WANCHO TONE TUP
- U+1E2ED WANCHO TONE TUPNI
- U+1E2EE WANCHO TONE KOI
- U+1E2EF WANCHO TONE KOINI

The four tone marks are in two pairs. One pair, WANCHO TONE TUP and WANCHO TONE TUPNI, is used with Southern Wancho, and the second pair, WANCHO TONE KOI and WANCHO TONE KOINI, is used with Northern Wancho.

Punctuation. Common Western punctuation marks such as comma, full stop, and question mark are used in Wancho.

Currency Sign. The Wancho currency sign, U+1E2FF WANCHO NGUN SIGN, is used to indicate rupees.

Digits. Wancho uses decimal digits 0–9 encoded in the range U+1E2F0..U+1E2F9. Common operators are used for mathematical operations.

13.19 Toto

13.19.1 Toto: U+1E290–U+1E2BF

The Toto script was created to write Toto, a Sino-Tibetan language spoken in West Bengal, India, near the Bhutan border. It was designed by Dhaniram Toto, a member of the Toto community, and officially launched in 2015. The language is also written with the Bengali/Bangla and Latin scripts.

Structure. Toto is a left-to-right alphabetic script comprised of letters representing consonants and vowels. There are five breathy vowels, which are separately encoded. The breathy vowels display a common distinguishing mark below the letter, such as is found in U+1E2A7 ҙ̖ TOTO LETTER BREATHY E. The distinguishing mark is not separately encoded; these characters are all treated atomically, and do not decompose. The mark below U+1E29C ҕ TOTO LETTER WA has a similar appearance, but does not indicate breathiness.

U+02BC ʼ MODIFIER LETTER APOSTROPHE is used for the glottal stop.

Tone. Toto has one tone mark, U+1E2AE ◌̇ TOTO SIGN RISING TONE, which appears only above vowels. The similar-shaped mark which appears above U+1E297 66 TOTO LETTER NA and U+1E298 6́ TOTO LETTER NGA is not a tone mark, and those letters do not decompose.

Punctuation and Digits. Common Western punctuation marks are currently used in Toto. The Toto script has no script-specific digits.

13.20 Tangsa

13.20.1 Tangsa: U+16A70–U+16ACF

Tangsa is an alphabetic script created by the late Lakhum Mossang in 1990 to write the Tangsa languages spoken in Arunachal Pradesh, India, and the Sagaing Region of northwest Myanmar. In January 2020 a few new characters (U+16A78..U+16A7B) were officially added to the script, including four characters for a short /a/-like sound.

In India the script is taught in some schools to write the local Muishvung (Mossang) variety of Tangsa. Some areas of India and Myanmar also use different Latin-based orthographies to write Tangsa languages.

Structure. The script is written left to right and contains 48 vowels and 31 consonants. Each vowel or diphthong contains an inherent tone, identified in the character name by an appended letter (Z, C, Q, X) used to mark tones in the Latin orthography developed by Reverend Gam Win. The tone letters indicate the tones numbered by linguists as 1, 3, 4, and 2, respectively. The Z, C, Q, X order reflects the order defined by the script's creator for the Muishvung variety of the language.

Punctuation. Western-style punctuation is employed in Tangsa.

Digits. Tangsa has script-specific decimal digits encoded in the range U+16AC0..U+16AC9.

13.21 Sunuwar

13.21.1 Sunuwar: U+11BC0–U+11BFF

The modern form of the Sunuwar script was developed by Karna Jentich in 1942, to write the Kiranti-Kõits (Sunuwar/Mukhia) language, which is spoken by around 40,000 people in Nepal and Sikkim (India). The script has been used in newspapers, government records, and other publications. In Nepal, the use of the script is promoted by the Sunuwar Welfare Society. Kiranti-Kõits has also been one of the official languages of Sikkim since 1996. In 2001 the Sikkim government mandated that state-run schools should offer instruction in both the primary language of the state and in a local language. Various additions and improvements have been made to the script, particularly around the turn of the century, and further changes to the orthography were proposed as recently as the early 2020s in Nepal.

Basic Features. Sunuwar text runs from left to right in horizontal lines. Words are separated by spaces, and the script is unicameral.

In Nepal the Sunuwar script is used as an alphabet, whereas in Sikkim it is used as an abugida. The repertoire for each is largely the same, but there are differences in usage and in a small number of letter forms.

Glyph Variants. There are regional variations between characters used in Sikkim and Nepal. For examples, see *Figure 13-12*.

Figure 13-12. Glyph Variants in Sunuwar

Character	Sikkim	Nepal
U+11BC5 SUNUWAR LETTER UTTHI	2 or ν	ν
U+11BC6 SUNUWAR LETTER KIK	℄	℄
U+11BCC SUNUWAR LETTER CARMI	⟨	⟨
U+11BD2 SUNUWAR LETTER SHYELE	ヲ	ヲ

Consonants. The Sunuwar block contains 27 consonant letters. Around the early 2000s the repertoire was augmented with new consonant letters to represent aspirated and retroflex sounds. However, use of the script is not yet fully standardized, and sometimes older ambiguous or digraph-based spellings still appear.

In Nepal, consonants have no inherent vowel and consonant clusters are simply indicated by groups of consonant letters. A lone consonant is not pronounced with a following vowel. In Sikkim, however, consonants have an inherent vowel, pronounced /ə/, and U+0331 ◌ COMBINING MACRON BELOW (called *sangmilu*) is

used to indicate suppression of the inherent vowel in clusters or when the letter stands alone. There are no conjuncts.

Onset consonant clusters with /-j/ or /-r/ follow a similar pattern. In Nepal these medial consonants are written using ordinary consonant letters, whereas in Sikkim a medial /-j/ is written using *sangmilu* followed by U+11BD4 𑯔 SUNUWAR LETTER YAT and medial /-r/ is written using U+032D ◌̭ COMBINING CIRCUMFLEX ACCENT BELOW (called *sangrums*).

Vowels. The Sunuwar block has six vowel letters that are all ordinary, spacing characters. Long vowel sounds can be indicated using a symbol called *laissi* after the vowel letter. U+003A ":" COLON is used for this. A colon may also be used as punctuation; if it occurs after a word ending with a vowel letter it will typically be preceded by a space in order to remove ambiguity.

Nasalization is indicated in Nepal using U+0303 ◌̃ COMBINING TILDE above a vowel letter. In Sikkim, nasalization is indicated instead with a flat line and dot above, called *taslathenk*. This mark is represented with U+0310 ◌̐ COMBINING CANDRABINDU, with a Sunuwar font-specific rendering. These conventions are illustrated on the left and right sides of <u>*Figure 13-13*</u>, respectively.

Figure 13-13. Sunuwar Nasalization in Nepal and Sikkim

𑯈𑯪̃:   /dẽ:/ "snail"   𑯈𑯪̐:

Tone Marks. A set of three combining marks are promoted in Nepal to represent Sunuwar tones. These are encoded as U+030D ◌̍ COMBINING VERTICAL LINE ABOVE, U+0301 ◌́ COMBINING ACUTE ACCENT, and U+0300 ◌̀ COMBINING GRAVE ACCENT. Over diphthongs the tone mark typically appears above the first vowel, and the nasalisation mark over the second, avoiding the need for stacked glyphs. In Sikkim, tone marks are not used.

Punctuation. U+11BE1 𑯡 SUNUWAR SIGN PVO represents an "auspicious syllable," which is uttered, often twice, before a formulaic phrase. The sign is written in salutations and benedictions, and its basic trident shape can vary in the details. It represents the sound /ɓ/ and is transcribed as *pvo*.

Other punctuation is the same as that used for the Latin script.

Numbers. A set of Sunuwar digits is encoded in the range U+10D40..U+10D49.

Line Breaking. Line breaking and justification are primarily based on inter-word spaces.

13.22 Gurung Khema

13.22.1 Gurung Khema: U+16100–U+1613F

The Gurung Khema script was developed by Bal Narsingh Gurung to write Gurung, a Tibeto-Burman language used in Nepal and the northeastern state of Sikkim in India. The script was introduced to the community in 1995 and revised several times, most recently in 2019. Another script for Gurung was created by Jagan Lal Gurung in 1944, called Khe Prih. Gurung is also written with the Devanagari, Tibetan, and Latin scripts.

Structure. Gurung Khema is an abugida based on Brahmic scripts, written from left to right. It makes use of a visible vowel killer, a nasalization mark, and four medial signs.

Vowels. The script uses vowel signs on consonants to modify the inherent vowel. The vowel signs can also be used on U+16100 ꄀ GURUNG KHEMA LETTER A to represent initial vowels. Originally Gurung Khema did not distinguish between short and long vowels, but four more vowel signs were added in 2019 to represent the long vowels. U+16129 ◌ GURUNG KHEMA VOWEL LENGTH MARK is required for decomposition of certain vowel signs, as shown in *Figure 13-14*. It is not used on its own.

Figure 13-14. Canonical Decomposition of Gurung Khema Vowel Signs

$$16121 \; \overset{=}{\circ} \; \textit{sign u} \;\; \equiv \;\; 1611E \, \overline{\circ} \; \textit{sign aa} + 1611E \, \overline{\circ} \; \textit{sign aa}$$

$$16122 \; \overset{=}{\circ} \; \textit{sign uu} \;\; \equiv \;\; 1611E \, \overline{\circ} \; \textit{sign aa} + 16129 \, \overset{\lrcorner}{\circ} \; \textit{length mark}$$

$$16123 \; \overset{\circ}{\circ} \; \textit{sign e} \;\; \equiv \;\; 1611E \, \overline{\circ} \; \textit{sign aa} + 1611F \, \overset{\circ}{\circ} \; \textit{sign i}$$

$$16124 \; \overset{\circ}{\circ} \; \textit{sign ee} \;\; \equiv \;\; 16129 \, \overset{\lrcorner}{\circ} \; \textit{length mark} + 1611F \, \overset{\circ}{\circ} \; \textit{sign i}$$

$$16125 \; \overset{\circ\circ}{\circ} \; \textit{sign ai} \;\; \equiv \;\; 1611E \, \overline{\circ} \; \textit{sign aa} + 16120 \, \overset{\circ\circ}{\circ} \; \textit{sign ii}$$

$$16126 \; \overset{\circ}{\circ} \; \textit{sign o} \;\; \equiv \;\; 16121 \, \overset{=}{\circ} \; \textit{sign u} + 1611F \, \overset{\circ}{\circ} \; \textit{sign i}$$

$$16127 \; \overset{\circ}{\circ} \; \textit{sign oo} \;\; \equiv \;\; 16122 \, \overset{=}{\circ} \; \textit{sign uu} + 1611F \, \overset{\circ}{\circ} \; \textit{sign i}$$

$$16128 \; \overset{\circ\circ}{\circ} \; \textit{sign au} \;\; \equiv \;\; 16121 \, \overset{=}{\circ} \; \textit{sign u} + 16120 \, \overset{\circ\circ}{\circ} \; \textit{sign ii}$$

Medial Signs. Medial signs imply the loss of the implicit vowel and follow consonants directly. A syllable can only contain a single instance of a *ya*, *va*, or *ra* medial sign. However, when a syllable contains any of those medial signs, it can optionally also be followed by the medial sign *ha*. When medial sign *ha* is used with another medial sign, *ha* always comes second in sequence, regardless of the implied phonetic order of the segments.

Other Signs. U+1612D ☋ GURUNG KHEMA SIGN ANUSVARA is used to indicate nasalization. U+1612F ♀ GURUNG KHEMA SIGN THOLHOMA represents the vowel killer. The vowel killer is always visible; the script does not form conjuncts. Both signs can be used in combination with the medial signs.

The mark for the *anusvara* is a small wedge shape that typically appears on top of vowel signs that have flat line glyphs, but underneath vowels signs consisting of a small circle or two circles. This means that it may also appear *between* parts of a vowel sign in the cases of complex vowels that have both circle and flat line parts in their glyphs. In all cases the *anusvara* is represented in the text directly after the vowel sign. The correct rendering of *anusvara* when combined with vowel signs is shown in <u>*Figure 13-15*</u>.

Figure 13-15. Gurung Khema Vowel Signs with *anusvara*

	本 *sign i*	本 *sign u*	本 *sign e*	本 *sign o*
本 *sign aa*	本 *sign ii*	本 *sign uu*	本 *sign ee*	本 *sign oo*
		本 *sign ai*	本 *sign au*	
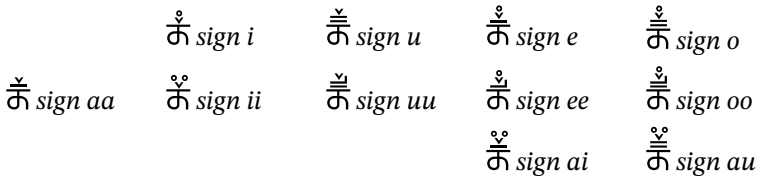
Punctuation. Gurung Khema uses Western-style punctuation along with U+0965 ‖ DEVANAGARI DOUBLE DANDA.

Digits. Gurung Khema script-specific digits are encoded in the range U+16130..U+16139.

13.23 Kirat Rai

13.23.1 Kirat Rai: U+16D40–U+16D7F

Kirat Rai is a script used to write the Bantawa language, spoken in eastern Nepal and the Indian states of Sikkim and West Bengal. The script is primarily used in India.

The most common name for the script is "Kirat Rai," however it is sometimes called "Khambu Rai Lipi" in West Bengal. The Kirat Rai script should not be confused with the Limbu script, which has sometimes been called Kirat Rai.

Structure. Kirat Rai is an abugida based on a simplified Brahmic model. It is written from left to right. The script does not have the rendering complexity of traditional Brahmic scripts (no reordering, no combining marks, and no conjuncts). Consonant letters have an inherent vowel, which is not pronounced in the final position. The script has no system of conjuncts but uses U+16D6B ▪ KIRAT RAI SIGN VIRAMA and U+16D6C ˏ KIRAT RAI SIGN SAAT to mute the inherent vowel. Vowel signs are not encoded as combining marks, but nevertheless must follow a consonant letter or the vowel carrier, U+16D43 ꜣ KIRAT RAI LETTER A.

Vowels. Eight vowel signs are encoded in the range U+16D63..U+16D6A. Three of these vowels can be visually analyzed as consisting of multiple parts corresponding to the shapes of other vowels: U+16D63 ꞁ KIRAT RAI VOWEL SIGN AA and U+16D67 ꞁ KIRAT RAI VOWEL SIGN E, as shown in <u>*Figure 13-16*</u>. These multipart vowels have been given canonical decompositions, so the atomic characters and the corresponding sequences are canonical equivalents. The atomic characters are the typical form when generating text.

Figure 13-16. Kirat Rai Multipart Vowels

U+16D68 ꞁꞁ KIRAT RAI VOWEL SIGN AI   ≡ 16D67 ꞁ + 16D67 ꞁ

U+16D69 ꞁꞁ KIRAT RAI VOWEL SIGN O   ≡ 16D63 ꞁ + 16D67 ꞁ

U+16D6A ꞁꞁꞁ KIRAT RAI VOWEL SIGN AU ≡ 16D69 ꞁꞁ + 16D67 ꞁ

Note that because canonical decompositions are applied recursively when normalizing Unicode text, the fully decomposed (NFD) form of U+16D6A ꞁꞁꞁ KIRAT RAI VOWEL SIGN AU is the three character sequence <16D63 ꞁ, 16D67 ꞁ, 16D67 ꞁ>.

Segmentation. Kirat Rai multipart vowels should not be broken apart when determining grapheme clusters. Structurally, they behave as if they were atomic elements, even if represented with a decomposed sequence. The Kirat Rai vowel letters have the property value Grapheme_Cluster_Break=V, which functions to keep the vowel letter sequences together when determining grapheme cluster boundaries.

Punctuation. Kirat Rai uses Western-style punctuation. U+002D "-" ʜʏᴘʜᴇɴ-
ᴍɪɴᴜs represents the *chamri* sign to form compound words, and U+16D6D ◦ ᴋɪʀᴀᴛ
ʀᴀɪ sɪɢɴ ʏᴜᴘɪ marks abbreviations. In addition, Kirat Rai uses two script-specific
dandas: U+16D6E / ᴋɪʀᴀᴛ ʀᴀɪ ᴅᴀɴᴅᴀ and U+16D6F // ᴋɪʀᴀᴛ ʀᴀɪ ᴅᴏᴜʙʟᴇ ᴅᴀɴᴅᴀ.

Digits. Kirat Rai script-specific digits are encoded in the range U+16D70..U+16D79.

Various Signs. U+16D40 ° ᴋɪʀᴀᴛ ʀᴀɪ sɪɢɴ ᴀɴᴜsᴠᴀʀᴀ denotes [n] and [ŋ]
phonemes. Nasalization of vowels is indicated by U+16D41 ᴋɪʀᴀᴛ ʀᴀɪ sɪɢɴ ᴛᴏɴᴘɪ.
U+16D42 ᴋɪʀᴀᴛ ʀᴀɪ sɪɢɴ ᴠɪsᴀʀɢᴀ represents the glottal stop phoneme.

Both U+16D6B ᴋɪʀᴀᴛ ʀᴀɪ sɪɢɴ ᴠɪʀᴀᴍᴀ and U+16D6C ᴋɪʀᴀᴛ ʀᴀɪ sɪɢɴ sᴀᴀᴛ are
used to mute the inherent vowel sound. *Saat* is only used to mute the inherent vowel
of the first letter of the word; all other places are represented by *virama*.

Chapter 14

South and Central Asia-III
Ancient Scripts

The oldest lengthy inscriptions of India, the edicts of Ashoka from the third century BCE, were written in two scripts, Kharoshthi and Brahmi. These are both ultimately of Semitic origin, probably deriving from Aramaic, which was an important administrative language of the Middle East at that time. Kharoshthi, which was written from right to left, was supplanted by Brahmi and its derivatives.

The Bhaiksuki script is a Brahmi-derived script used around 1000 CE, primarily in the area of the present-day states of Bihar and West Bengal in India and northern Bangladesh. Surviving Bhaiksuki texts are limited to a few Buddhist manuscripts and inscriptions.

Phags-pa is an historical script related to Tibetan that was created as the national script of the Mongol empire. Phags-pa was used mostly in Eastern and Central Asia for writing text in the Mongolian and Chinese languages.

The Marchen script (Tibetan *sMar-chen*) is a Brahmi-derived script used in the Tibetan Bön liturgical tradition. Marchen is used to write Tibetan and the historic Zhang-zhung language. Although few historical examples of the script have been found, Marchen appears in modern-day inscriptions and in modern Bön literature.

The Old Turkic script is known from eighth-century Siberian stone inscriptions, and is the oldest known form of writing for a Turkic language. Also referred to as Turkic Runes due to its superficial resemblance to Germanic Runes, it appears to have evolved from the Sogdian script, which is in turn derived from Aramaic.

Both the Soyombo script and the Zanabazar Square script are historic scripts used to write Mongolian, Sanskrit, and Tibetan. These two scripts were both invented by Zanabazar (1635–1723), one of the most important Buddhist leaders in Mongolia. Each script is an *abugida*. Soyombo appears primarily in Buddhist texts in Central Asia. Zanabazar Square has also been called "Horizontal Square" script, "Mongolian Horizontal Square" script and "Xewtee Dörböljin Bicig."

Old Sogdian and Sogdian are related scripts used in Central Asia. The Old Sogdian script was used for a group of related writing systems dating from the third to the sixth century CE. These writing systems were all used to write Sogdian, an eastern

Iranian language. Old Sogdian is a non-joining abjad. Its basic repertoire consists of 20 of the 22 letters of the Aramaic alphabet.

The Sogdian script, which derives from Old Sogdian, is also an abjad, and was used from the seventh to the fourteenth century CE, also to write Sogdian. Its repertoire corresponds to that of Old Sogdian, but has a number of differences in the glyphs and also has additional characters. The script was also used to write Chinese, Sanskrit, and Uyghur. Sogdian is the ancestor of the Old Uyghur and Mongolian scripts.

The Old Uyghur script flourished between the 8th and 17th centuries in northwest China and other parts of Asia. Originally used to write medieval Turkish languages, its use later expanded to write other languages, including Chinese, Mongolian, Tibetan and Arabic. Old Uyghur is a cursive joining alphabet, and developed from the cursive style of the Sogdian script. The default orientation of the script is horizontal, with the script being read from right to left.

14.1 Brahmi

14.1.1 Brahmi: U+11000–U+1107F

The Brahmi script is an historical script of India attested from the third century BCE until the late first millennium CE. Over the centuries Brahmi developed many regional varieties, which ultimately became the modern Indian writing systems, including Devanagari, Tamil and so on. The encoding of the Brahmi script in the Unicode Standard supports the representation of texts in Indian languages from this historical period. For texts written in historically transitional scripts—that is, between Brahmi and its modern derivatives—there may be alternative choices to represent the text. In some cases, there may be a separate encoding for a regional medieval script, whose use would be appropriate. In other cases, users should consider whether the use of Brahmi or a particular modern script best suits their needs.

Encoding Model. The Brahmi script is an *abugida* and is encoded using the Unicode *virama* model. Consonants have an inherent vowel /a/. A separate character is encoded for the virama: U+11046 BRAHMI VIRAMA. The *virama* is used between consonants to form conjunct consonants. It is also used as an explicit killer to indicate a vowelless consonant.

Vowel Letters. Vowel letters are encoded atomically in Brahmi, even if they can be analyzed visually as consisting of multiple parts. *Table 14-1* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 14-1. Brahmi Vowel Letters

For	Use	Do Not Use
ᴚ	11006	<11005, 11038>
ᴕ	1100C	<1100B, 1103E>
ᴧ	11010	<1100F, 11042>

Rendering Behavior. Consonant conjuncts are represented by a sequence including virama: <C, virama, C>. In Brahmi these consonant conjuncts are rendered as consonant ligatures. Up to a very late date, Brahmi used vertical conjuncts exclusively, in which the ligation involves stacking of the consonant glyphs vertically. The Brahmi script does not have a parallel series of half-consonants, as developed in Devanagari and some other modern Indic scripts.

The elements of consonant ligatures are laid out from top left to bottom right, as shown for *sva* in *Figure 14-1*. Preconsonantal *r*, postconsonantal *r* and postconsonantal *y* assume special reduced shapes in all except the earliest varieties of Brahmi.

The *kṣa* and *jña* ligatures, however, are often transparent, as also shown in <u>*Figure 14-1*</u>.

Figure 14-1. Consonant Ligatures in Brahmi
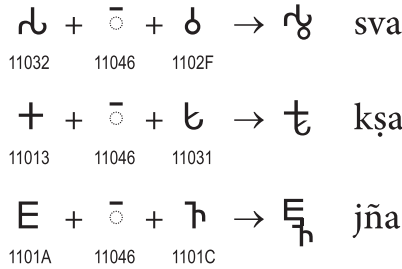
A vowelless consonant is represented in text by following the consonant with a *virama*: <C, virama>. The presence of the *virama* "kills" the vowel. Such vowelless consonants have visible distinctions from regular consonants, and are rendered in one of two major styles. In the first style, the vowelless consonant is written smaller and lower than regular consonants, and often has a connecting line drawn from the vowelless consonant to the preceding *aksara*. In the second style, a horizontal line is drawn above the vowelless consonant. The second style is the basis for the representative glyph for U+11046 BRAHMI VIRAMA in the code charts. These differences in presentation are purely stylistic; it is up to the font developers and rendering systems to render Brahmi vowelless consonants in the appropriate style.

Vowel Modifiers. U+11000 BRAHMI SIGN CANDRABINDU indicates nasalization of a vowel. U+11001 BRAHMI SIGN ANUSVARA is used to indicate that a vowel is nasalized (when the next syllable starts with a fricative), or that it is followed by a nasal segment (when the next syllable starts with a stop). U+11002 BRAHMI SIGN VISARGA is used to write syllable-final voiceless /h/. The velar and labial allophones of /h/ (that is, [x] and [ɸ], followed by voiceless velar and labial stops respectively) are sometimes written with separate signs U+11003 BRAHMI SIGN JIHVAMULIYA and U+11004 BRAHMI SIGN UPADHMANIYA. Unlike *visarga*, these two signs have the properties of a letter, and are not considered combining marks. They enter into ligatures with the following homorganic voiceless stop consonant, without the use of a *virama*.

Old Tamil Brahmi. Brahmi was used to write the Tamil language starting from the second century BCE. The different orthographies used to write Tamil in the Brahmi script are covered by the Unicode encoding of Brahmi.

In one of these orthographies the inherent vowel of Brahmi consonant letters is dropped, and U+11038 BRAHMI VOWEL SIGN AA is used to represent both short and long [a] / [a:]. In this orthography, consonant signs without a vowel sign always represent the bare consonant without an inherent vowel.

Some orthographies employ U+11070 BRAHMI SIGN OLD TAMIL VIRAMA to cancel the inherent vowel of the consonants, but the virama does not form conjuncts. The glyph for Old Tamil virama is a dot, called a *puḷḷi*, which may appear identical to U+11001 BRAHMI SIGN ANUSVARA. Fonts may differentiate the Old Tamil virama from the Brahmi *anusvara* by placing the dots at different positions according to the style of the font. These orthographies also use Old Tamil short vowels [e] and [o], which are atomically encoded at U+11071..U+11074. The glyphs for these vowels appear with a *puḷḷi*, but the short vowels [e] and [o] are not decomposed.

Distinct Old Tamil consonants not found in Prakrit and Sanskrit are encoded at U+11035..U+11037 and U+11075. When U+1102B BRAHMI LETTER MA occurs in Old Tamil text, it may be shown with a glyphic variant distinct from the form shown in the Brahmi code charts.

Bhattiprolu Brahmi. Ten short Middle Indo-Aryan inscriptions from the second century BCE found at Bhattiprolu in Andhra Pradesh show an orthography that seems to be derived from the Tamil Brahmi system. To avoid the phonetic ambiguity of the Tamil Brahmi U+11038 BRAHMI VOWEL SIGN AA (standing for either [a] or [a:]), the Bhattiprolu inscriptions introduced a separate vowel sign for long [a:] by adding a vertical stroke to the end of the earlier sign. This is encoded as U+11039 BRAHMI VOWEL SIGN BHATTIPROLU AA.

Punctuation. There are seven punctuation marks in the encoded repertoire for Brahmi. The single and double dandas, U+11047 BRAHMI DANDA and U+11048 BRAHMI DOUBLE DANDA, delimit clauses and verses. U+11049 BRAHMI PUNCTUATION DOT, U+1104A BRAHMI PUNCTUATION DOUBLE DOT, and U+1104B BRAHMI PUNCTUATION LINE delimit smaller textual units, while U+1104C BRAHMI PUNCTUATION CRESCENT BAR and U+1104D BRAHMI PUNCTUATION LOTUS separate larger textual units.

Line Breaking. Line breaks may occur after every orthographic syllable.

Numerals. Two sets of numbers, used for different numbering systems, are attested in Brahmi documents. The first set is the old additive-multiplicative system that goes back to the beginning of the Brahmi script. The second is a set of ten decimal digits that occurs side by side with the earlier numbering system in manuscripts and inscriptions during the late Brahmi period.

The set of additive-multiplicative numerals of the Brahmi script contains separate signs for the digits from 1 to 9, the tens from 10 to 90, as well as signs for 100 and 1000. Numbers are written additively, with the higher-valued signs preceding the lower-valued ones. Multiples of 100 and of 1000 are expressed multiplicatively with character sequences consisting of the sign for 100 or 1000, followed by U+1107F BRAHMI NUMBER JOINER and then the multiplier. The component parts of additive numbers are rendered unligated, whereas multiples are rendered in ligated form.

For example, the sequence <U+11064 BRAHMI NUMBER ONE HUNDRED, U+11055 BRAHMI NUMBER FOUR> represents the number 100 + 4 = 104 and is rendered unligated, whereas the sequence <U+11064 BRAHMI NUMBER ONE HUNDRED, U+1107F BRAHMI NUMBER JOINER, U+11055 BRAHMI NUMBER FOUR> represents the number 100 × 4 = 400 and is rendered as a ligature.

U+1107F BRAHMI NUMBER JOINER forms a ligature between the two numeral characters surrounding it. It functions similarly to U+2D7F TIFINAGH CONSONANT JOINER, but is intended to be used only with Brahmi numerals in the range U+11052 BRAHMI NUMBER ONE through U+11065 BRAHMI NUMBER ONE THOUSAND, and not with consonants or other characters. Because U+1107F BRAHMI NUMBER JOINER marks a semantic distinction between additive numbers and multiples, it should be rendered with a visible fallback glyph to indicate its presence in the text when it cannot be displayed by normal rendering.

In addition to the ligated forms of the multiples of 100 and 1000, other examples from the middle and late Brahmi periods show the signs for 200, 300, and 2000 in special forms not obviously connected with a ligature of the component parts. Such forms may be enabled in fonts using a ligature substitution.

A special sign for zero was invented later, and the positional system came into use. This system is the ancestor of modern decimal number systems. Due to the different systemic features and shapes, the signs in this set are separately encoded in the range from U+11066 BRAHMI DIGIT ZERO through U+1106F BRAHMI DIGIT NINE. These signs have the same properties as the modern Indic digits. Examples are shown in *Table 14-2*. Brahmi decimal digits are categorized as regular bases and can act as vowel carriers, whereas the numerals U+11052 BRAHMI NUMBER ONE through U+11065 BRAHMI NUMBER ONE THOUSAND and their ligatures formed with U+1107F BRAHMI NUMBER JOINER are not used as vowel carriers.

Table 14-2. Brahmi Positional Digits

Display	Value	Code Points
·	0	11066
╲	1	11067
२	2	11068
३	3	11069
⟩	4	1106A
╲·	10	<11067, 11066>
२३⟩	234	<11068, 11069, 1106A>

14.2 Kharoshthi

14.2.1 Kharoshthi: U+10A00–U+10A5F

The Kharoshthi script, properly spelled as Kharoṣṭhī, was used historically to write Gāndhārī and Sanskrit as well as various mixed dialects. Kharoshthi is an Indic script of the *abugida* type. However, unlike other Indic scripts, it is written from right to left. The Kharoshthi script was initially deciphered around the middle of the 19th century by James Prinsep and others who worked from short Greek and Kharoshthi inscriptions on the coins of the Indo-Greek and Indo-Scythian kings. The decipherment has been refined over the last 150 years as more material has come to light.

The Kharoshthi script is one of the two ancient writing systems of India. Unlike the pan-Indian Brāhmī script, Kharoshthi was confined to the northwest of India centered on the region of *Gandhāra* (modern northern Pakistan and eastern Afghanistan, as shown in *Figure 14-2*). Gandhara proper is shown on the map as the dark gray area near Peshawar. The lighter gray areas represent places where the Kharoshthi script was used and where manuscripts and inscriptions have been found.

Figure 14-2. Geographical Extent of the Kharoshthi Script

The exact details of the origin of the Kharoshthi script remain obscure, but it is almost certainly related to Aramaic. The Kharoṣṭhī script first appears in a fully developed form in the Aśokan inscriptions at Shahbazgarhi and Mansehra which have been dated to around 250 BCE. The script continued to be used in Gandhara and neighboring regions, sometimes alongside Brahmi, until around the third century CE, when it disappeared from its homeland. Kharoshthi was also used for official docu-

ments and epigraphs in the Central Asian cities of Khotan and Niya in the third and fourth centuries CE, and it appears to have survived in Kucha and neighboring areas along the Northern Silk Road until the seventh century. The Central Asian form of the script used during these later centuries is termed *Formal Kharoshthi* and was used to write both Gandhari and Tocharian B. Representation of Kharoshthi in the Unicode code charts uses forms based on manuscripts of the first century CE.

Directionality. Kharoshthi can be implemented using the rules of the Unicode Bidirectional Algorithm. Both letters and digits are written from right to left. Kharoshthi letters do not have positional variants.

Diacritical Marks and Vowels. All vowels other than *a* are written with diacritical marks in Kharoshthi. In addition, there are six vowel modifiers and three consonant modifiers that are written with combining diacritics. In general, only one combining vowel sign is applied to each syllable (*aksara*). However, there are some examples of two vowel signs on *aksaras* in the Kharoshthi of Central Asia.

Numerals. Kharoshthi employs a set of eight numeral signs unique to the script. Like the letters, the numerals are written from right to left. Numbers in Kharoshthi are based on an additive system. There is no zero, nor separate signs for the numbers five through nine. The number 1996, for example, would logically be represented as 1000 4 4 1 100 20 20 20 20 10 4 2 and would appear as shown in *Figure 14-3*. The numerals are encoded in the range U+10A40..U+10A47.

Figure 14-3. Kharoshthi Number 1996

Punctuation. Nine different punctuation marks are used in manuscripts and inscriptions. The punctuation marks are encoded in the range U+10A50..U+10A58.

Word Breaks, Line Breaks, and Hyphenation. Most Kharoshthi manuscripts are written as continuous text with no indication of word boundaries. Only a few examples are known where spaces have been used to separate words or verse quarters. Most scribes tried to finish a word before starting a new line. There are no examples of anything akin to hyphenation in Kharoshthi manuscripts. In cases where a word would not completely fit into a line, its continuation appears at the start of the next line. Modern scholarly practice uses spaces and hyphenation. When necessary, hyphenation should follow Sanskrit practice.

Sorting. There is an ancient ordering connected with Kharoshthi called *Arapacana*, named after the first five *aksaras*. However, there is no evidence that words were sorted in this order, and there is no record of the complete *Arapacana* sequence. In modern scholarly practice, Gandhari is sorted in much the same order as Sanskrit. Vowel length, even when marked, is ignored when sorting Kharoshthi.

14.2.2 Rendering Kharoshthi

Rendering requirements for Kharoshthi are similar to those for Devanagari. This section specifies a minimum set of combining rules that provide legible Kharoshthi diacritic and ligature substitution behavior.

All unmarked consonants include the inherent vowel *a*. Other vowels are indicated by one of the combining vowel diacritics. Some letters may take more than one diacritical mark. In these cases the preferred sequence is Letter + {Consonant Modifier} + {Vowel Sign} + {Vowel Modifier}. For example the Sanskrit word *parārdhyaiḥ* might be rendered in Kharoshthi script as *parārjaiḥ, written from right to left, as shown in *Figure 14-4*.

Figure 14-4. Kharoshthi Rendering Example
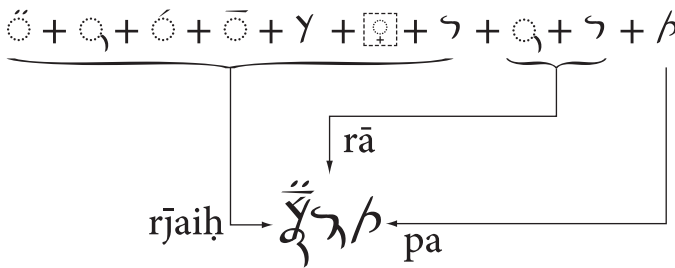
Combining Vowels. The various combining vowels attach to characters in different ways. A number of groupings have been determined on the basis of their visual types, such as horizontal or vertical, as shown in *Table 14-3*.

Table 14-3. Kharoshthi Vowel Signs

Type	Example	Group Members
Vowel sign i		
Horizontal	a + -i → i	A, NA, HA
Vertical	tha + -i → thi	THA, PA, PHA, MA, LA, SHA
Diagonal	ka + -i → ki	All other letters
Vowel sign u		
Independent	ha + -u → hu	TTA, HA
Ligated	ma + -u → mu	MA

Attached	a + -u → u ୨ + ୰ → ୨	All other letters
Vowel sign vocalic r		
Attached	a + -ṛ → ṛ ୨ + ୯ → ୨	A, KA, KKA, KHA, GA, GHA, CA, CHA, JA, TA, DA, DHA, NA, PA, PHA, BA, BHA, VA, SHA, SA
Independent	ma + -ṛ → mṛ ꞏ + ୯ → ꞏ	MA, HA
Vowel sign e		
Horizontal	a + -e → e ୨ + ó → ୨	A, NA, HA
Vertical	tha + -e → the 𝆑 + ó → 𝆑	THA, PA, PHA, LA, SSA
Ligated	da + -e → de ꮥ + ó → ꮯ	DA, MA
Diagonal	ka + -e → ke ꑥ + ó → ꑥ	All other letters
Vowel sign o		
Vertical	pa + -o → po 𝆑 + ୰ → 𝆑	PA, PHA, YA, SHA
Diagonal	a + -o → o ୨ + ୰ → ୨	All other letters

Combining Vowel Modifiers. U+10A0C ୰ KHAROSHTHI VOWEL LENGTH MARK indicates equivalent long vowels and, when used in combination with -e and -o, indicates the diphthongs –*ai* and –*au*. U+10A0D ୰ KHAROSHTHI SIGN DOUBLE RING BELOW appears in some Central Asian documents, but its precise phonetic value has not yet been established. These two modifiers have been found only in manuscripts and inscriptions from the first century CE onward. U+10A0E ୰ KHAROSHTHI SIGN ANUSVARA indicates nasalization, and U+10A0F ö KHAROSHTHI SIGN VISARGA is generally used to indicate unvoiced syllable-final [h], but has a secondary use as a vowel length marker. *Visarga* is found only in Sanskritized forms of the language and is not known to occur in a single *aksara* with *anusvara*. The modifiers and the vowels they modify are given in *Table 14-4*.

Table 14-4. Kharoshthi Vowel Modifiers

Type	Example	Group Members
Vowel length mark	ma + ō → mā ꞏ + ୯ → ꞏ	A, I, U, R, E, O
Double ring below	sa + ୰ → sạ ୨ + ୰ → ꝫ	A, U
Anusvara	a + -ṃ → aṃ ୨ + ୰ → ୨	A, I, U, R, E, O

Visarga	ka + -ḥ → kaḥ 𐨐 + ◌ → 𐨐	A, I, U, R, E, O

Combining Consonant Modifiers. U+10A38 ◌ KHAROSHTHI SIGN BAR ABOVE indicates various modified pronunciations depending on the consonants involved, such as nasalization or aspiration. U+10A39 ◌ KHAROSHTHI SIGN CAUDA indicates various modified pronunciations of consonants, particularly fricativization. The precise value of U+10A3A ◌ KHAROSHTHI SIGN DOT BELOW has not yet been determined. Usually only one consonant modifier can be applied to a single consonant. The resulting combined form may also combine with vowel diacritics, one of the vowel modifiers, or anusvara or visarga. The modifiers and the consonants they modify are given in *Table 14-5*.

Table 14-5. Kharoshthi Consonant Modifiers

Type	Example	Group Members
Bar above	ja + ◌ → j̄a 𐨗 + ◌ → 𐨗	GA, CA, JA, NA, MA, SHA, SSA, SA, HA
Cauda	ga + ◌ → ǵa 𐨒 + ◌ → 𐨒	GA, JA, DDA, TA, DA, PA, YA, VA, SHA, SA
Dot below	ma + ◌ → ṃa 𐨨 + ◌ → 𐨨	MA, HA

Virama. The virama is used to indicate the suppression of the inherent vowel. The glyph for U+10A3F ◌ KHAROSHTHI VIRAMA shown in the code charts is arbitrary and is not actually rendered directly; the dotted box around the glyph indicates that special rendering is required. When not followed by a consonant, the virama causes the preceding consonant to be written as subscript to the left of the letter preceding it. If followed by another consonant, the virama will trigger a combined form consisting of two or more consonants. The resulting form may also be subject to combinations with the previously noted combining diacritics.

The virama can follow only a consonant or a consonant modifier. It cannot follow a space, a vowel, a vowel modifier, a number, a punctuation sign, or another virama. Examples of the use of the Kharoshthi virama are given in *Table 14-6*.

Table 14-6. Examples of Kharoshthi Virama

Type	Example
Pure virama	dha + i + k + virama → dhik 𐨢 + ◌ + 𐨐 + ◌ → 𐨢
Ligatures	ka + virama + ṣa → kṣa 𐨐 + ◌ + 𐨮 → 𐨐
Consonants with special combining forms	sa + virama + ya → sya 𐨯 + ◌ + 𐨩 → 𐨯

Consonants with full combined form	ka + virama + ta → kta ꜣ + ◌ + ꜗ → ꝫ

Subjoined ya. A special form of subjoined *ya* appears in the Kharoshthi documents from Niya. In most cases this sign occurs in loan words into Gandhari. The most common source for these loans is presumed to be Tocharian A, where the sequence -*ly*- is normal. This special form resembles the full form of *ya* (∧), attached cursively to the stem of the preceding consonant sign. This contrasts with the common form of subjoined *ya* which is a looped flourish extension of the stem. The special form of *ya* can be requested using U+200D ZERO WIDTH JOINER as shown in *Figure 14-5*.

Figure 14-5. Subjoined Forms of *ya*

la + virama + ya → lya ㇳ+ ◌ + ∧ → ⅎ
la + ZWJ + virama + ya → lýa ㇳ+ ᶻʷᴶ+ ◌ + ∧ → ⅄

14.3 Bhaiksuki

14.3.1 Bhaiksuki: U+11C00–U+11C6F

The Bhaiksuki script is a Brahmi-derived script used from about the 10th to the 13th centuries CE, primarily in the area of the present-day states of Bihar and West Bengal in India and northern Bangladesh. The original name of the script was Saindhavī (that is, the Sindhu or Indus script), but after its discovery in the late 19th century, scholars called it Bhaiksuki or they used a descriptive name, the Arrow-headed script. The script is used to write the Middle Indic language, which is also called Saindhavī. Surviving Bhaiksuki texts are limited to a few Buddhist manuscripts and inscriptions.

Structure. The structure of Bhaiksuki script is similar to that of other Brahmi-based Indic scripts. It is an *abugida* that makes use of a virama. The script is written from left to right.

Rendering. Many of the vowel signs have contextual variants when they occur with certain consonants. The consonants U+11C22 BHAIKSUKI LETTER PA, U+11C27 BHAIKSUKI LETTER YA, and U+11C28 BHAIKSUKI LETTER RA have special combining forms when they occur with certain vowel signs.

Virama and Conjuncts. The script includes a virama, U+11C3F BHAIKSUKI SIGN VIRAMA, which functions to suppress the inherent vowel and to form conjuncts. Consonant clusters are generally rendered as vertically stacked ligatures, with non-initial consonants attached below the initial letter. Above-base vowel signs and consonant letters attach to the glyph of the initial consonant, while below-base vowel signs attach to the glyph of the final consonant. The letters *ka*, *pa*, *ra,* and *ya* take special forms when they occur in conjuncts.

The Bhaiksuki dependent vowel signs in the range U+11C38..U+11C3B, *e*, *ai*, *o*, and *au*, are simply treated as above-base vowel signs. Although the historically cognate vowel signs may be treated as having left-side parts, or as two- or three-part vowels in many other scripts of India, the peculiarities of rendering for these vowel signs in the Bhaiksuki script can be handled more easily with the above-base designations. The dependent vowel signs *ai*, *o*, and *au* are not given formal canonical decompositions, but are encoded instead as atomic characters.

The sequence <C, *virama*> is rendered using a visible virama by default. The combinations <*ta*, *virama*>, <*na*, *virama*>, and <*ma*, *virama*> may also be displayed with special ligatures; there is no apparent semantic distinction between sequences containing the visible virama and sequences displayed as ligatures.

Various Signs. Nasalization is represented by U+11C3C BHAIKSUKI SIGN CANDRA-BINDU and U+11C3D BHAIKSUKI SIGN ANUSVARA. Post-vocalic aspiration in Sanskrit is indicated by U+11C3E BHAIKSUKI SIGN VISARGA. Use of U+11C40 BHAIKSUKI SIGN AVAGRAHA indicates elision of a word-initial *a* in Sanskrit as a result of sandhi.

Digits and Numbers. Bhaiksuki has a script-specific set of decimal digits. Because the glyphs for zero and three have not been yet identified in the Bhaiksuki corpus, representative glyphs for U+11C50 BHAIKSUKI DIGIT ZERO and U+11C53 BHAIKSUKI DIGIT THREE are based upon corresponding digits in other scripts that are contemporaneous with Bhaiksuki.

In addition to the decimal digits, the script has a distinct numerical notation system. Bhaiksuki contains numbers for primary and tens units, and U+11C6C BHAIKSUKI HUNDREDS UNIT MARK. The numbers are written vertically, with the largest number written above smaller units. Control of vertical orientation is managed at the font level, but the default rendering is horizontal left to right.

Punctuation. The script employs script-specific dandas, U+11C41 BHAIKSUKI DANDA and U+11C42 BHAIKSUKI DOUBLE DANDA. Words are separated by U+11C43 BHAIKSUKI WORD SEPARATOR. Two characters, U+11C44 BHAIKSUKI GAP FILLER-1 and U+11C45 BHAIKSUKI GAP FILLER-2, are used as spacing or completion marks, especially to indicate the end of a line. They also can indicate a deliberate elision or an otherwise missing portion of text.

14.4 Phags-pa

14.4.1 Phags-pa: U+A840–U+A87F

The Phags-pa script is an historic script with some limited modern use. It bears some similarity to Tibetan and has no case distinctions. It is written vertically in columns running from left to right, like Mongolian. Units are often composed of several syllables and may be separated by whitespace.

The term *Phags-pa* is often written with an initial apostrophe: '*Phags-pa*. The Unicode Standard makes use of the alternative spelling without an initial apostrophe because apostrophes are not allowed in the normative character and block names.

History. The Phags-pa script was devised by the Tibetan lama Blo-gros rGyal-mtshan [lodoi jaltsan] (1235–1280 CE), commonly known by the title *Phags-pa Lama* ("exalted monk"), at the behest of Khubilai Khan (reigned 1260–1294) when he assumed leadership of the Mongol tribes in 1260. In 1269, the "new Mongolian script," as it was called, was promulgated by imperial edict for use as the national script of the Mongol empire, which from 1279 to 1368, as the Yuan dynasty, encompassed all of China.

The new script was not only intended to replace the Uyghur-derived script that had been used to write Mongolian since the time of Genghis Khan (reigned 1206–1227), but was also intended to be used to write all the diverse languages spoken throughout the empire. Although the Phags-pa script never succeeded in replacing the earlier Mongolian script and had only very limited usage in writing languages other than Mongolian and Chinese, it was used quite extensively during the Yuan dynasty for a variety of purposes. There are many monumental inscriptions and manuscript copies of imperial edicts written in Mongolian or Chinese using the Phags-pa script. The script can also be found on a wide range of artifacts, including seals, official passes, coins, and banknotes. It was even used for engraving the inscriptions on Christian tombstones. A number of books are known to have been printed in the Phags-pa script, but all that has survived are some fragments from a printed edition of the Mongolian translation of a religious treatise by the Phags-pa Lama's uncle, Sakya Pandita. Of particular interest to scholars of Chinese historical linguistics is a rhyming dictionary of Chinese with phonetic readings for Chinese ideographs given in the Phags-pa script.

An ornate, pseudo-archaic "seal script" version of the Phags-pa script was developed specifically for engraving inscriptions on seals. The letters of the seal script form of Phags-pa mimic the labyrinthine strokes of Chinese seal script characters. A great many official seals and seal impressions from the Yuan dynasty are known. The seal script was also sometimes used for carving the title inscription on stone stelae, but never for writing ordinary running text.

Although the vast majority of extant Phags-pa texts and inscriptions from the thirteenth and fourteenth centuries are written in the Mongolian or Chinese languages, there are also examples of the script being used for writing Uyghur, Tibetan, and Sanskrit, including two long Buddhist inscriptions in Sanskrit carved in 1345.

After the fall of the Yuan dynasty in 1368, the Phags-pa script was no longer used for writing Chinese or Mongolian. However, the script continued to be used on a limited scale in Tibet for special purposes such as engraving seals. By the late sixteenth century, a distinctive, stylized variety of Phags-pa script had developed in Tibet, and this Tibetan-style Phags-pa script, known as *hor-yig,* "Mongolian writing" in Tibetan, is still used today as a decorative script. In addition to being used for engraving seals, the Tibetan-style Phags-pa script is used for writing book titles on the covers of traditional style books, for architectural inscriptions such as those found on temple columns and doorways, and for calligraphic samplers.

Basic Structure. The Phags-pa script is based on Tibetan, but unlike any other Brahmic script Phags-pa is written vertically from top to bottom in columns advancing from left to right across the writing surface. This unusual directionality is borrowed from Mongolian, as is the way in which Phags-pa letters are ligated together along a vertical stem axis. In modern contexts, when embedded in horizontally oriented scripts, short sections of Phags-pa text may be laid out horizontally from left to right.

Despite the difference in directionality, the Phags-pa script fundamentally follows the Tibetan model of writing, and consonant letters have an inherent /a/ vowel sound. However, Phags-pa vowels are independent letters, not vowel signs as is the case with Tibetan, so they may start a syllable without being attached to a null consonant. Nevertheless, a null consonant (U+A85D PHAGS-PA LETTER A) is still needed to write an initial /a/ and is orthographically required before a diphthong or the semivowel U+A867 PHAGS-PA SUBJOINED LETTER WA. Only when writing Tibetan in the Phags-pa script is the null consonant required before an initial pure vowel sound.

Except for the *candrabindu* (which is discussed later in this section), Phags-pa letters read from top to bottom in logical order, so the vowel letters *i*, *e*, and *o* are placed below the preceding consonant—unlike in Tibetan, where they are placed above the consonant they modify.

Syllable Division. Text written in the Phags-pa script is broken into discrete syllabic units separated by whitespace. When used for writing Chinese, each Phags-pa syllabic unit corresponds to a single Han ideograph. For Mongolian and other polysyllabic languages, a single word is typically written as several syllabic units, each separated from each other by whitespace.

For example, the Mongolian word *tengri,* "heaven," which is written as a single ligated unit in the Mongolian script, is written as two separate syllabic units, *deng ri*,

in the Phags-pa script. Syllable division does not necessarily correspond directly to grammatical structure. For instance, the Mongolian word *usun,* "water," is written *u sun* in the Phags-pa script, but its genitive form *usunu* is written *u su nu.*

Within a single syllabic unit, the Phags-pa letters are normally ligated together. Most letters ligate along a righthand stem axis, although reversed-form letters may instead ligate along a lefthand stem axis. The letter U+A861 PHAGS-PA LETTER O ligates along a central stem axis.

In traditional Phags-pa texts, normally no distinction is made between the whitespace used in between syllables belonging to the same word and the whitespace used in between syllables belonging to different words. Line breaks may occur between any syllable, regardless of word status. In contrast, in modern contexts, influenced by practices used in the processing of Mongolian text, U+202F NARROW NO-BREAK SPACE (NNBSP) may be used to separate syllables within a word, whereas U+0020 SPACE is used between words—and line breaking would be affected accordingly.

Candrabindu. U+A873 PHAGS-PA LETTER CANDRABINDU is used in writing Sanskrit mantras, where it represents a final nasal sound. However, although it represents the final sound in a syllable unit, it is always written as the first glyph in the sequence of letters, above the initial consonant or vowel of the syllable, but not ligated to the following letter. For example, *om* is written as a *candrabindu* followed by the letter *o.* To simplify cursor placement, text selection, and so on, the *candrabindu* is encoded in visual order rather than logical order. Thus *om* would be represented by the sequence <U+A873, U+A861>, rendered as shown in <u>*Figure 14-6*</u>.

Figure 14-6. Phags-pa Syllable Om

As the *candrabindu* is separated from the following letter, it does not take part in the shaping behavior of the syllable unit. Thus, in the syllable *om*, the letter *o* (U+A861) takes the isolate positional form.

Alternate Letters. Four alternate forms of the letters *ya*, *sha*, *ha,* and *fa* are encoded for use in writing Chinese under certain circumstances:

U+A86D PHAGS-PA LETTER ALTERNATE YA

U+A86E PHAGS-PA LETTER VOICELESS SHA

U+A86F PHAGS-PA LETTER VOICED HA

U+A870 PHAGS-PA LETTER ASPIRATED FA

These letters are used in the early-fourteenth-century Phags-pa rhyming dictionary of Chinese, *Menggu ziyun*, to represent historical phonetic differences between Chinese

syllables that were no longer reflected in the contemporary Chinese language. This dictionary follows the standard phonetic classification of Chinese syllables into 36 initials, but as these had been defined many centuries previously, by the fourteenth century some of the initials had merged together or diverged into separate sounds. To distinguish historical phonetic characteristics, the dictionary uses two slightly different forms of the letters *ya*, *sha*, *ha,* and *fa*.

The historical phonetic values that U+A86E, U+A86F, and U+A870 represent are indicated by their character names, but this is not the case for U+A86D, so there may be some confusion as to when to use U+A857 PHAGS-PA LETTER YA and when to use U+A86D PHAGS-PA LETTER ALTERNATE YA. U+A857 is used to represent historic null initials, whereas U+A86D is used to represent historic palatal initials.

Numbers. There are no special characters for numbers in the Phags-pa script, so numbers are spelled out in full in the appropriate language.

Punctuation. The vast majority of traditional Phags-pa texts do not make use of any punctuation marks. However, some Mongolian inscriptions borrow the Mongolian punctuation marks U+1802 MONGOLIAN COMMA, U+1803 MONGOLIAN FULL STOP, and U+1805 MONGOLIAN FOUR DOTS.

Additionally, a small circle punctuation mark is used in some printed Phags-pa texts. This mark can be represented by U+3002 IDEOGRAPHIC FULL STOP, but for Phags-pa the *ideographic full stop* should be centered, not positioned to one side of the column. This follows traditional, historic practice for rendering the ideographic full stop in Chinese text, rather than more modern typography.

Tibetan Phags-pa texts also use head marks, U+A874 PHAGS-PA SINGLE HEAD MARK U+A875 PHAGS-PA DOUBLE HEAD MARK, to mark the start of an inscription, and *shad* marks, U+A876 PHAGS-PA MARK SHAD and U+A877 PHAGS-PA MARK DOUBLE SHAD, to mark the end of a section of text.

Positional Variants. The four vowel letters U+A85E PHAGS-PA LETTER I, U+A85F PHAGS-PA LETTER U, U+A860 PHAGS-PA LETTER E, and U+A861 PHAGS-PA LETTER O have different isolate, initial, medial, and final glyph forms depending on whether they are immediately preceded or followed by another Phags-pa letter (other than U+A873 PHAGS-PA LETTER CANDRABINDU, which does not affect the shaping of adjacent letters). The code charts show these four characters in their isolate form. The various positional forms of these letters are shown in .

Table 14-7. Phags-pa Positional Forms of I, U, E, and O

Letter	Isolate	Initial	Medial	Final
U+A85E PHAGS-PA LETTER I	ꡞ	ꡞ	ꡞ	ꡞ
U+A85F PHAGS-PA LETTER U	ꡟ	ꡟ	ꡟ	ꡟ

U+A860 PHAGS-PA LETTER E	ㄱ	ㄱ	ㄱ	ㄱ
U+A861 PHAGS-PA LETTER O	ㅈ	ㅈ	↑	ㅅ

Consonant letters and the vowel letter U+A866 PHAGS-PA LETTER EE do not have distinct positional forms, although initial, medial, final, and isolate forms of these letters may be distinguished by the presence or absence of a stem extender that is used to ligate to the following letter.

The invisible format characters U+200D ZERO WIDTH JOINER (ZWJ) and U+200C ZERO WIDTH NON-JOINER (ZWNJ) may be used to override the expected shaping behavior, in the same way that they do for Mongolian and other scripts (see *Chapter 23, Special Areas and Format Characters*). For example, ZWJ may be used to select the initial, medial, or final form of a letter in isolation:

    <U+200D, U+A861, U+200D> selects the medial form of PHAGS-PA LETTER O

    <U+200D, U+A861> selects the final form of PHAGS-PA LETTER O

    <U+A861, U+200D> selects the initial form of PHAGS-PA LETTER O

Conversely, ZWNJ may be used to inhibit expected shaping. For example, in the sequence <U+A85E, U+200C, U+A85F, U+200C, U+A860, U+200C, U+A861> there will be no cursive joining between the Phags-pa letters *i*, *u*, *e*, and *o*.

Mirrored Variants. The four characters U+A869 PHAGS-PA LETTER TTA, U+A86A PHAGS-PA LETTER TTHA, U+A86B PHAGS-PA LETTER DDA, and U+A86C PHAGS-PA LETTER NNA are mirrored forms of the letters U+A848 PHAGS-PA LETTER TA, U+A849 PHAGS-PA LETTER THA, U+A84A PHAGS-PA LETTER DA, and U+A84B PHAGS-PA LETTER NA, respectively, and are used to represent the Sanskrit retroflex dental series of letters. Because these letters are mirrored, their stem axis is on the lefthand side rather than the righthand side, as is the case for all other consonant letters. This means that when the letters *tta*, *ttha*, *dda*, and *nna* occur at the start of a syllable unit, to correctly ligate with them any following letters normally take a mirrored glyph form. Because only a limited number of words use these letters, only the letters U+A856 PHAGS-PA LETTER SMALL A, U+A85C PHAGS-PA LETTER HA, U+A85E PHAGS-PA LETTER I, U+A85F PHAGS-PA LETTER U, U+A860 PHAGS-PA LETTER E, and U+A868 PHAGS-PA SUBJOINED LETTER YA are affected by this glyph mirroring behavior. The Sanskrit syllables that exhibit glyph mirroring after *tta*, *ttha*, *dda*, and *nna* are shown in *Table 14-8*.

Table 14-8. Contextual Glyph Mirroring in Phags-pa

Character	Syllables with Glyph Mirroring	Syllables without Glyph Mirroring
U+A856 PHAGS-PA LETTER SMALL A	*tthā*	*ttā, tthā*

U+A85E PHAGS-PA LETTER I	*tthi, nni*	*tthi*
U+A85F PHAGS-PA LETTER U	*nnu*	
U+A860 PHAGS-PA LETTER E	*tthe, dde, nne*	
U+A85C PHAGS-PA LETTER HA	*ddha*	
U+A868 PHAGS-PA SUBJOINED LETTER YA	*nnya*	

Glyph mirroring is not consistently applied to the letters U+A856 PHAGS-PA LETTER SMALL A and U+A85E PHAGS-PA LETTER I in the extant Sanskrit Phags-pa inscriptions. The letter *i* may occur both mirrored and unmirrored after the letter *ttha*, although it always occurs mirrored after the letter *nna*. *Small a* is not normally mirrored after the letters *tta* and *ttha* as its mirrored glyph is identical in shape to U+A85A PHAGS-PA LETTER SHA. Nevertheless, *small a* does sometimes occur in a mirrored form after the letter *ttha*, in which case context indicates that this is a mirrored letter *small a* and not the letter *sha*.

When any of the letters *small a*, *i*, *u*, *e*, *ha*, or *subjoined ya* immediately follow either *tta*, *ttha*, *dda*, or *nna* directly or another mirrored letter, then a mirrored glyph form of the letter should be selected automatically by the rendering system. Although *small a* is not normally mirrored in extant inscriptions, for consistency it is mirrored by default after *tta*, *ttha*, *dda*, and *nna* in the rendering model for Phags-pa.

To override the default mirroring behavior of the letters *small a*, *ha*, *i*, *u*, *e*, and *subjoined ya*, U+FE00 VARIATION SELECTOR-1 (VS1) may be applied to the appropriate character, as shown in *Table 14-9*. Note that only the variation sequences shown in *Table 14-9* are valid; any other sequence of a Phags-pa letter and VS1 is unspecified.

Table 14-9. Phags-pa Standardized Variants

Character Sequence	Description of Variant Appearance
<U+A856, U+FE00>	*phags-pa letter reversed shaping small a*
<U+A85C, U+FE00>	*phags-pa letter reversed shaping ha*
<U+A85E, U+FE00>	*phags-pa letter reversed shaping i*
<U+A85F, U+FE00>	*phags-pa letter reversed shaping u*
<U+A860, U+FE00>	*phags-pa letter reversed shaping e*
<U+A868, U+FE00>	*phags-pa letter reversed shaping ya*

In *Table 14-9*, "reversed shaping" means that the appearance of the character is reversed with respect to its expected appearance. Thus, if no mirroring would be expected for the character in the given context, applying VS1 would cause the rendering engine to select a mirrored glyph form. Similarly, if context would dictate glyph mirroring, application of VS1 would inhibit the expected glyph mirroring. This mechanism will typically be used to select a mirrored glyph for the letters *small a*, *ha*, *i*,*u*, *e*, or *subjoined ya* in isolation (for example, in discussion of the Phags-pa script)

or to inhibit mirroring of the letters *small a* and *i* when they are not mirrored after the letters *tta* and *ttha*, as shown in <u>*Figure 14-7*</u>.

Figure 14-7. Phags-pa Reversed Shaping

The first example illustrates the normal shaping for the syllable *thi*. The second example shows the reversed shaping for *i* in that syllable and would be represented by a standardized variation sequence: <U+A849, U+A85E, U+FE00>. Example 3 illustrates the normal shaping for the Sanskrit syllable *tthi*, where the reversal of the glyph for the letter *i* is automatically conditioned by the lefthand stem placement of the Sanskrit letter *ttha*. Example 4 shows reversed shaping for *i* in the syllable *tthi* and would be represented by a standardized variation sequence: <U+A86A, U+A85E, U+FE00>.

Cursive Joining. Joining types are defined for Phags-pa characters in the file Arabic-Shaping.txt. Joining types identify the joining behavior of characters in cursive joining scripts and were originally introduced for the Arabic script. Because the Phags-pa script is typically rendered from top to bottom, Joining_Type = L (Left_Joining) conventionally refers to bottom joining that is, joining to a character which follows (is below) it. Joining_Type = R (Right_Joining) is not used for the Phags-pa script, but would refer to top joining, that is, joining to a character which precedes (is above) it. Most Phags-pa characters are Dual_Joining, as they may join on both top and bottom.

The L and R designations of the Joining_Type property should not be confused with the left-hand and right-hand placement of stem axes in the Phags-pa script in vertical layout. Whether a Phags-pa character joins on the left-hand or right-hand side in its stem axis is not defined in ArabicShaping.txt.

14.5 Marchen

14.5.1 Marchen: U+11C70–U+11CBF

The Marchen script (Tibetan *sMar-chen*) is a Brahmi-derived script used in the Tibetan Bön liturgical tradition. Marchen is used to write Tibetan and also the historic Zhang-zhung language. The script is said to originate in the ancient kingdom of Zhang-zhung, which flourished in western and northern Tibet before Buddhism was introduced in the area in the seventh century. Although few historical examples of the script have been found, Marchen appears in modern-day inscriptions and is widely used in modern Bön literature.

Encoding Model. The encoding model for Marchen follows that of Tibetan. Marchen contains thirty base consonants and thirty subjoined consonants, which can be used to form vertical stacks of two or more consonants. Although not all subjoined consonants have been identified in extant texts, the full set of subjoined forms is encoded, so that all possible stack combinations can be represented.

Vowels and Consonants. As in Tibetan, two or more Marchen consonants can stack vertically. Vowel signs are placed above, below, or alongside a stack of one or more consonants.

Other Signs. Marchen includes a vowel lengthener, U+11CB0 MARCHEN VOWEL SIGN AA, known as *a-chung*. Nasalization is represented by U+11CB6 MARCHEN SIGN CANDRABINDU and U+11CB5 MARCHEN SIGN ANUSVARA.

Punctuation. There are two script-specific punctuation marks encoded. U+11C70 MARCHEN HEAD MARK corresponds to U+0F04 TIBETAN MARK INITIAL YIG MGO MDUN MA. The sentence-final shad mark, U+11C71 MARCHEN MARK SHAD, corresponds to U+0F0D TIBETAN MARK SHAD. Marchen does not use an explicit mark to separate syllables; this differs from the use of the Tibetan *tsek* (tsheg) mark.

14.6 Zanabazar Square

14.6.1 Zanabazar Square: U+11A00–U+11A4F

The Zanabazar Square script is an *abugida* based upon Tibetan and inspired by the Brahmi model. The script has some similarities with both Tibetan and Phags-pa. It was used to write Mongolian, Sanskrit, and Tibetan, and has also been called "Horizontal Square" script, "Mongolian Horizontal Square" script and "Xewtee Dörböljin Bicig."

The script was invented by Zanabazar (1635–1723), one of the most important Buddhist leaders in Mongolia, who also developed the Soyombo script. Its creation likely preceded that of Soyombo.

Structure. The Zanabazar Square script is written from left to right. The script is generally written horizontally, but in some instances occurs in vertical environments. Consonant letters possess the inherent vowel /a/.

The phonetic value of a consonant letter is changed by the attachment of a vowel sign. In Mongolian, the inherent vowel is suppressed by a final-consonant mark, which indicates both a syllable-final consonant and a syllabic boundary. In Sanskrit or Tibetan, the virama silences the inherent vowel of a consonant, but does not mark syllable boundaries.

Vowels and Diphthongs. The Zanabazar Square script has one vowel letter, nine dependent vowel marks, and one vowel length mark. The *letter a* vowel, U+11A00 ZANABAZAR SQUARE LETTER A, has the value /a/ when it occurs independently. It can also assume the value of a combined vowel sign.

A long vowel is represented by placing the vowel length mark, U+11A0A ZANABAZAR SQUARE VOWEL LENGTH MARK, after a consonant or vowel sign. When combined with the *letter a* vowel or a consonant letter, the length mark lengthens the inherent vowel /a/ to /ā/. Vowel signs are used with the *letter a* vowel and with consonants. Multiple vowel signs may combine with a single base letter. Independent vowels are represented by attaching vowel signs to the *letter a* vowel , except for U+11A09 ZANABAZAR SQUARE VOWEL SIGN REVERSED I. The vowel sign *reversed i* is used for writing four Sanskrit vocalic letters.

U+11A07 ZANABAZAR SQUARE VOWEL SIGN AI and U+11A08 ZANABAZAR SQUARE VOWEL SIGN AU represent the diphthongs *ai* and *au*. They also function as secondary vowel signs for *i* and *u* to produce additional diphthongs in Mongolian.

Consonants. There are 40 consonants, including the following:

- U+11A26 ZANABAZAR SQUARE LETTER DZHA represents Sanskrit *jha*

- U+11A29 ZANABAZAR SQUARE LETTER -A represents Tibetan *'a chung*
- U+11A32 ZANABAZAR SQUARE LETTER KSSA represents Sanskrit cluster *kṣa* (/kṣa/)

Consonant clusters are written as conjuncts, which are generally rendered as vertical stacks, with each non-initial letter subjoined sequentially beneath the initial letter of the cluster.

The consonants *ya*, *ra*, *la*, *va* have different representations when they occur in Sanskrit and Tibetan conjuncts. Therefore, contextual forms of these letters are encoded as separate characters.

Virama and Subjoiner. U+11A34 ZANABAZAR SQUARE SIGN VIRAMA is used to silence the inherent vowel of a consonant for writing Sanskrit and Tibetan. The virama is used only with a consonant and behaves as other combining marks in the script, always with a visible display.

Vowel-silencing characters in Brahmi-based scripts often have a secondary function of controlling conjunct formation, however, the Zanabazar Square script does not follow this pattern. A separate character, U+11A47 ZANABAZAR SQUARE SUBJOINER, is used to control conjunct formation.

The representation of a vertical conjunct stack uses the *subjoiner* character between each consonant of the cluster. For example, the syllable *mstu* is represented with the sequence <ma, subjoiner, sa, subjoiner, ta, vowel sign ue>, as shown in the second line of *Figure 14-8*. To suppress the visual stacking of a cluster, the *virama* character is used instead, which kills the vowel and results in a visual marking of the dead consonant which does not stack. For example, if the syllable *mstu* is represented with the sequence <ma, virama, sa, virama, ta, vowel sign ue>, the rendering is as shown in the first row of *Figure 14-8*.

Figure 14-8. Conjunct Stacking in Zanabazar Square
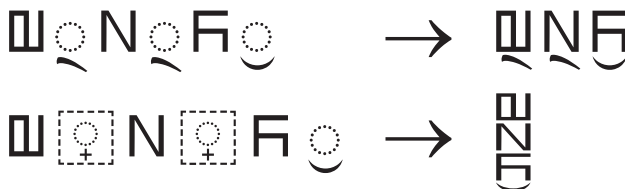
Head Marks. There are four head marks in the Zanabazar Square script. These four head marks are used in transliterations of Tibetan texts when written with the Zanabazar Square script. They occur at the beginning of texts.

- U+11A3F ZANABAZAR SQUARE INITIAL HEAD MARK

- U+11A40 ZANABAZAR SQUARE CLOSING HEAD MARK

- U+11A45 ZANABAZAR SQUARE INITIAL DOUBLE-LINED HEAD MARK

- U+11A46 ZANABAZAR SQUARE CLOSING DOUBLE-LINED HEAD MARK

Both U+11A3F ZANABAZAR SQUARE INITIAL HEAD MARK and U+11A45 ZANABAZAR SQUARE INITIAL DOUBLE-LINED HEAD MARK are used as a base for candrabindu and anusvara signs.

The U+11A40 ZANABAZAR SQUARE CLOSING HEAD MARK and U+11A46 ZANABAZAR SQUARE CLOSING DOUBLE-LINED HEAD MARK may be used for producing extended head marks, similar to usage in Tibetan.

Other Marks. Two vowel modifiers are used to transliterate words of Sanskrit origin:

- U+11A38 ZANABAZAR SQUARE SIGN ANUSVARA indicates nasalization

- U+11A39 ZANABAZAR SQUARE SIGN VISARGA indicates post-vocalic aspiration

In addition, three combining signs are used as nasalization marks and ornaments for the head mark:

- U+11A35 ZANABAZAR SQUARE SIGN CANDRABINDU

- U+11A36 ZANABAZAR SQUARE SIGN CANDRABINDU WITH ORNAMENT

- U+11A37 ZANABAZAR SQUARE SIGN CANDRA WITH ORNAMENT

The U+11A33 ZANABAZAR SQUARE FINAL CONSONANT MARK marks syllable-final consonants when writing Mongolian.

Numerals. There are no known script-specific numerals.

Punctuation. The Zanabazar Square script includes four punctuation marks used for writing Tibetan:

- U+11A41 ZANABAZAR SQUARE MARK TSHEG indicates the end of a syllable

- U+11A42 ZANABAZAR SQUARE MARK SHAD indicates the end of the phrase or sentence

- U+11A43 ZANABAZAR SQUARE MARK DOUBLE SHAD marks the end of a text section

- U+11A44 ZANABAZAR SQUARE MARK LONG TSHEG behaves as a comma

14.7 Soyombo

14.7.1 Soyombo: U+11A50–U+11AAF

The Soyombo script is an historic script used to write Mongolian, Sanskrit, and Tibetan. It was created in 1686 by Zanabazar (1635–1723), who also developed the Zanabazar Square script. The script appears primarily in Buddhist texts in Central Asia. Most of these texts consist of either handwritten manuscripts or inscriptions.

Structure. Soyombo is an *abugida*. Consonants generally include an inherent vowel /a/, as is the case with many other Brahmi-derived scripts. The script also includes final consonant signs and four cluster-initial letters. A special subjoiner is employed to create conjuncts.

Soyombo text is typically written horizontally from left to right. In vertically written text, characters are oriented in columns laid out from left to right, with upright glyphs.

The graphical structure of Soyombo letters consists of two parts: a frame, made up of a vertical bar with a triangle at the top, and a nucleus that represents a phoneme. Together the frame and the nucleus represent the atomic letter. Vowel signs, final consonants, and other phonetic features appear as dependent signs attached to the letters. The signs may appear above or to the right of the frame, or below the nucleus.

Vowels and Diphthongs. The vowel *a* is represented by U+11A50 SOYOMBO LETTER A. When it occurs with a vowel sign, SOYOMBO LETTER A serves as a vowel-carrier, indicating an independent vowel. Long vowels are represented by appending U+11A5B SOYOMBO VOWEL LENGTH MARK. When used to write Mongolian, U+11A57 SOYOMBO VOWEL SIGN AI and U+11A58 SOYOMBO VOWEL SIGN AU are used with other vowel signs to represent diphthongs.

Consonants. Mongolian syllable-final consonants are represented by U+11A50 SOYOMBO LETTER A followed by a final consonant sign. To indicate geminated consonants, U+11A98 SOYOMBO GEMINATION MARK is stacked above the triangle of the frame. In the backing store, it occurs immediately after the base letter, but before any other combining mark. Other above-base signs are shown above the gemination mark.

Generally, consonant clusters are written as a conjunct forms. Because Soyombo does not have a native virama, a special subjoiner character, U+11A99 SOYOMBO SUBJOINER, is used. Conjuncts are represented by using a subjoiner between each pair of consonants in a cluster. A conjunct is rendered as a vertical stack of the regular form of the initial letter and the nucleus of each non-initial letter. Four cluster-initial letters have special forms: *la*, *sha*, *sa* and *ra*. Depending upon the

context, clusters involving these four letters may be rendered using the stacked or prefixed forms. The consonant cluster *kssa* has the structure of an atomic letter, and is separately encoded as U+11A83 SOYOMBO LETTER KSSA.

Character Names. The character names are based on their values for writing Tibetan, with the exception of the final consonant signs, which reflect their Mongolian usage. The order of the consonant letters follows the alphabetical order of the Tibetan script. This also matches the order of letters in the Zanabazar Square script.

Other Marks. Two vowel modifiers are used to transliterate words of Sanskrit origin, U+11A96 SOYOMBO SIGN ANUSVARA, which indicates nasalization, and U+11A97 SOYOMBO SIGN VISARGA, which is used to indicate post-vocalic aspiration. Independent forms of these modifiers are represented by combining them with U+11A50 SOYOMBO LETTER A.

Numerals. There are no known script-specific numerals.

Punctuation. The Soyombo script includes a number of punctuation marks. U+11A9A SOYOMBO MARK TSHEG indicates the end of a syllable, and corresponds to U+0F0B TIBETAN MARK INTERSYLLABIC TSHEG. To indicate the end of a phrase or syllable, U+11A9B SOYOMBO MARK SHAD may be employed. It corresponds to U+0F0D TIBETAN MARK SHAD and U+0964 DEVANAGARI DANDA. The end of a section is marked by U+11A9C SOYOMBO MARK DOUBLE SHAD, corresponding to U+0F0E TIBETAN MARK NYIS SHAD and U+0965 DEVANAGARI DOUBLE DANDA.

The script also contains three head marks, similar to those used in Mongolian and Tibetan. The Soyombo marks may be followed by a *shad* or *double shad*. The U+11A9E SOYOMBO HEAD MARK WITH MOON AND SUN AND TRIPLE FLAME, also known as the Svayambhu or "Soyombo" sign, is the official symbol of Mongolia. In addition, the script includes terminal marks, which appear at the end of text.

14.8 Old Turkic

14.8.1 Old Turkic: U+10C00–U+10C4F

The origins of the Old Turkic script are unclear, but it seems to have evolved from a non-cursive form of the Sogdian script, one of the Aramaic-derived scripts used to write Iranian languages, in order to write the Old Turkish language. Old Turkic is attested in stone inscriptions from the early eighth century CE found around the Orkhon River in Mongolia, and in a slightly different version in stone inscriptions of the later eighth century found in Siberia near the Yenisei River and elsewhere. These inscriptions are the earliest written examples of a Turkic language. By the ninth century the Old Turkic script had been supplanted by the Uyghur script.

Because Old Turkic characters superficially resemble Germanic runes, the script is also known as Turkic Runes and Turkic Runiform, in addition to the names Orkhon script, Yenisei script, and Siberian script.

Where the Orkhon and Yenisei versions of a given Old Turkic letter differ significantly, each is separately encoded.

Structure. Old Turkish vowels can be classified into two groups based on their front or back articulation. A given word uses vowels from only one of these groups; the group is indicated by the form of the consonants in the word, because most consonants have separate forms to match the two vowel types. Other phonetic rules permit prediction of rounded and unrounded vowels, and high, medium or low vowels within a word. Some consonants also indicate that the preceding vowel is a high vowel. Thus, most initial and medial vowels are not explicitly written; only vowels that end a word are always written, and there is sometimes ambiguity about whether a vowel precedes a given consonant.

Ligature. Old Turkic includes one ligature, which is used to represent [tʃɪ]. It should be represented as:

$$\text{λ} \quad + \quad \boxed{\text{ZW}\atop\text{J}} \quad + \quad \ulcorner \quad \rightarrow \quad \text{λ}$$

10C32   200D   10C03

Directionality. For horizontal writing, the Old Turkic script is written from right to left within a row, with rows running from bottom to top. Conformant implementations of Old Turkic script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

In some cases, under Chinese influence, the layout was rotated ninety degrees counterclockwise to produce vertical columns of text in which the characters are read top to bottom within a column, and the columns are read right to left.

Punctuation. Word division and some other punctuation functions are usually indi-
cated by a two-dot mark similar to a colon; U+205A TWO DOT PUNCTUATION may be
used to represent this punctuation mark. In some cases a mark such as U+2E30 RING
POINT is used instead.

14.9 Old Sogdian

14.9.1 Old Sogdian: U+10F00–U+10F2F

The Old Sogdian script is used to represent a group of related writing systems of Central Asia dating from the third to the sixth century CE. These writing systems were all used to write Sogdian, an eastern Iranian language. Old Sogdian is based on four sets of written materials: the Kultobe inscriptions in modern Kazakhstan; the preserved epistles called the "Ancient Letters," which are the earliest attested Sogdian manuscripts found in Dunhuang, China; inscriptions from the Upper Indus area of Pakistan; and inscriptions found on coins and vessels around Tashkent, Uzbekistan.

Repertoire. The basic repertoire consists of 20 of the 22 letters of the Aramaic alphabet. However, some of the original Aramaic letters ceased to be distinct in Old Sogdian. In the Ancient Letters, the usual glyph for *resh* is identical to the glyph for *daleth* and for *ayin*. As a result, *resh*, *ayin* and *daleth* are unified as a single character, U+10F18 OLD SOGDIAN LETTER RESH-AYIN-DALETH. In addition, the Old Sogdian repertoire includes six final letters, three final letters with vertical tail, and one alternate letter, U+10F13 OLD SOGDIAN LETTER ALTERNATE AYIN. The script also includes one heterogram, U+10F27 OLD SOGDIAN LIGATURE AYIN-DALETH, meaning "to," used in salutations in the Ancient Letters.

Structure. Old Sogdian is a non-joining *abjad*, like Hebrew. The letters retain their shape within a word, and six letters, *aleph*, *beth*, *he*, *nun*, *sadhe*, and *taw,* have distinctive word-final forms. Adjacent letters may connect or overlap due to cursive writing, but unlike the later Sogdian script, letters do not change their shape based on word position.

Orientation. Most Old Sogdian text is written right to left, in lines running from top to bottom. Some Upper Indus inscriptions are written vertically, with the letters rotated ninety degrees counter-clockwise, in columns running from left to right. As a result of this behavior in vertical writing, Old Sogdian characters are given the Vertical_Orientation property value R.

Numbers. Ten Sogdian-specific numbers and fractions are encoded in the range U+10F1D..U+10F26.

Punctuation. No script-specific punctuation marks have been attested.

14.10 Sogdian

14.10.1 Sogdian: U+10F30–U+10F6F

Derived from Old Sogdian, the Sogdian script was used from the seventh to the four-teenth century CE in Central Asia to write the eastern Iranian language Sogdian. It was also used to write Chinese, Sanskrit, and Uyghur. Sogdian is the ancestor of the Mongolian and Old Uyghur scripts. It is attested in manuscripts and inscribed on coins, stone, pottery, and other media. The script has two major styles: "formal," used in Buddhist *sutra* manuscripts, and a simplified, "cursive" style. The Old Uyghur script is believed to have derived from the Sogdian cursive style in the eighth or ninth century CE.

Structure. Sogdian is an *abjad* that can be written horizontally from right to left, or vertically from top to bottom, in columns running from left to right. When the script appears in vertical orientation, the glyphs are rotated ninety degrees counter-clock-wise. Unlike Old Sogdian, Sogdian is a cursive joining script. Eleven combining signs in the range U+10F46..U+10F50 are used for disambiguation and transcription.

The Sogdian repertoire corresponds to that of Old Sogdian, but has a number of differences in the glyphs and also has additional characters. Sogdian has a special form of *ayin* for an Aramaic heterogram, and includes two characters not found in Old Sogdian, *feth* and *lesh*. The letter *feth* is used to represent [f]. *Lesh* or "hooked resh" is an extension of *resh-ayin* with a below-base hook that has become an intrinsic part of the letter. The repertoire includes one phonogram, U+10F45 SOGDIAN INDEPENDENT SHIN, an alternate form of isolated *shin*, used to transcribe one Chinese character, U+6240 所. The glyph for *ayin* is identical to the glyph for *resh*; therefore the two letters have been unified as a single character, U+10F40 SOGDIAN LETTER RESH-AYIN.

Glyphs. The representative glyphs are generally based on the isolated or independent form of letters found in the formal style of Sogdian. Fonts may be used to show the formal or cursive style of a text. As in other *abjads*, the letters connect and change shape based on their position within a word. In the later Sogdian styles, some letters, such as *nun*, *gimel* and *beth*, remain unconnected from a following letter to distinguish them from similar shapes.

Numbers. The Sogdian script includes script-specific numbers encoded in the range U+10F51..U+10F54.

Punctuation. Five script-specific punctuation characters are included in the reper-toire. The four Sogdian punctuation characters, U+10F55 SOGDIAN PUNCTUATION TWO VERTICAL BARS, U+10F56 SOGDIAN PUNCTUATION TWO VERTICAL BARS WITH DOTS, U+10F57 SOGDIAN PUNCTUATION CIRCLE WITH DOT and U+10F58 SOGDIAN

PUNCTUATION TWO CIRCLES WITH DOTS, delimit text segments and may vary in shape. U+10F59 SOGDIAN PUNCTUATION HALF CIRCLE WITH DOT generally indicates the completion of a text. Various other punctuation marks occur in Sogdian texts, and in some cases may be represented by punctuation characters from other blocks, such as General Punctuation.

14.11 Old Uyghur

14.11.1 Old Uyghur: U+10F70–U+10FAF

The historical Old Uyghur script flourished between the 8th and 17th centuries, primarily in the Tarim Basin of northwest China and other parts of Asia. The script was originally used to write medieval Turkish languages, but was later expanded to write other languages, including Chinese, Mongolian, Tibetan and Arabic. Old Uyghur developed from the cursive style of the Sogdian script (see *Section 14.10, Sogdian*) and is the ancestor of the Mongolian script (see *Section 13.5, Mongolian*).

The script has two main styles. "Square" style is a formal, book style where the letters are carefully written out. The square style is found in manuscripts, official documents, and in block printing for religious and literary texts. The second main style is "cursive," used for rapid writing, particularly for administrative documents, as well as religious and literary texts. Other styles also developed, such as "post-Mongolic," which was employed for literary and civil documents after the 14th century.

Structure. Old Uyghur is a cursive joining alphabet. The default orientation of the script is horizontal, with the script being read from right to left. Although the script is traditionally laid out vertically in columns that run left to right, horizontal orientation facilitates the handling of Old Uyghur in multilingual contexts. Texts with vertical orientation should be handled by vertical text layout.

Repertoire. Based on evidence from 9th century documents, the Old Uyghur repertoire contained 15 consonants and three additional letters—*aleph*, *waw* and *yodh*—used to mark long vowels. The letters *aleph*, *waw* and *yodh* combine as digraphs and trigraphs to represent vowels of the Turkic languages.

Over time, some Old Uyghur letters fell together. For example, in the 11th century *samekh* and *shin* were both represented by *shin*. Diacritics were used to distinguish the merged letters: *samekh* was written using U+10F7F OLD UYGHUR LETTER SHIN, and *shin* was written with <U+10F7F OLD UYGHUR LETTER SHIN, U+10F85 OLD UYGHUR COMBINING TWO DOTS BELOW>. The reading of Old Uyghur text may be ambiguous due to the merger of letters and the nature of rapid, cursive writing. This ambiguity can be addressed using markup.

Representative Glyphs. The representative glyphs are based on the isolated form of the square style letters. Contextual forms of the letters are based on normalized shapes of the square style and from block prints. The square and cursive styles are not encoded separately. Fonts should handle the different styles, which can vary across regions and time. The terminals of many Old Uyghur letters, such as *aleph* and *beth*, may have different orientations and should be treated as glyph variants.

Shaping Behavior. Most Old Uyghur characters are dual-joining, except *zayin* and *heth*, which are right-joining.

Punctuation. U+10F86 ᴏʟᴅ ᴜʏɢʜᴜʀ ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ ʙᴀʀ and U+10F87 ᴏʟᴅ ᴜʏɢʜᴜʀ ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ ᴛᴡᴏ ʙᴀʀs delimit text sections for shorter and longer sections, respectively. In a similar way, U+10F88 ᴏʟᴅ ᴜʏɢʜᴜʀ ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ ᴛᴡᴏ ᴅᴏᴛs separates shorter text units and U+10F89 ᴏʟᴅ ᴜʏɢʜᴜʀ ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ ꜰᴏᴜʀ ᴅᴏᴛs separates longer sections. The script also uses a sign that is unified with U+10AF2 ᴍᴀɴɪᴄʜᴀᴇᴀɴ ᴘᴜɴᴄᴛᴜᴀᴛɪᴏɴ ᴅᴏᴜʙʟᴇ ᴅᴏᴛ ᴡɪᴛʜɪɴ ᴅᴏᴛ.

Word boundaries are indicated by spaces. In documents with the square script, letters with extended horizontal terminals may be stretched to touch the initial letter of the following word. However, this behavior reflects no semantic distinction, and in plain text spaces should be used between words. To represent the joining of the two words calligraphically, U+200C ᴢᴇʀᴏ ᴡɪᴅᴛʜ ɴᴏɴ-ᴊᴏɪɴᴇʀ may be used. Some texts extend the initial baseline to fill out the space on a line. For example, the space between the last word in the line and the margin may be filled by using U+0640 ᴀʀᴀʙɪᴄ ᴛᴀᴛᴡᴇᴇʟ between the last two letters of a word.

Other Signs. Four diacritics with dots encoded in the range U+10F82..U+10F85 differentiate merged letters and indicate sounds for which no distinct letter exists. The diacritics, whose shapes may vary across different script styles, commonly occur with *nun*, *gimel*, *heth*, *samekh*, and *shin*. No script-specific digits have been encoded.

Chapter 15

South and Central Asia-IV
Other Historic Scripts

This chapter documents other modern and historic scripts of South and Central Asia.

Most of these scripts are historically related to the other scripts of India, and most are ultimately derived from the Brahmi script. None of them were standardized in ISCII. The encoding for each script is done on its own terms, and the blocks do not make use of a common pattern for the layout of code points.

This introduction briefly identifies each script, occasionally highlighting the most salient distinctive attributes of the script. Details are provided in the individual block descriptions that follow.

Syloti Nagri is used to write the modern Sylheti language of northeast Bangladesh and southeast Assam in India.

Kaithi is a historic North Indian script, closely related to the Devanagari and Gujarati scripts. It was used in the area of the present-day states of Bihar and Uttar Pradesh in northern India, from the 16th century until the early 20th century.

Sharada is a historical script that was used to write Sanskrit, Kashmiri, and other languages of northern South Asia; it was the principal inscriptional and literary script of Kashmir from the 8th century CE until the 20th century. It has limited and specialized modern use.

Takri, descended from Sharada, is used in northern India and surrounding countries. It is the traditional writing system for the Chambeali and Dogri languages, as well as several "Pahari" languages. In addition to popular usage for commercial and informal purposes, Takri served as the official script of several princely states of northern and northwestern India from the 17th century until the middle of the 20th century.

During the 17th century, the Brahmi-based Dogra script was used to write the Dogri language in Jammu and Kashmir in the northern region of the Indian subcontinent. The Dogra script was standardized in the 1860s, and is closely related to the Takri script. Dogri is now usually written with the Devanagari script.

Siddham is another Brahmi-based writing system related to Sharada, and structurally similar to Devanagari. It originated in India, and was used across South, Central, and East Asia, and is presently predominantly used in East Asia. Originally used for writing Buddhist manuscripts, the script is still used by Japanese Buddhist communities.

Mahajani is a Brahmi-based alphabet commonly used by bankers and money lenders across northern India until the middle of the 20th century. It is a specialized commercial script used for writing accounts and financial records. Mahajani has similarities to Landa, Kaithi, and Devanagari.

Khojki is a writing system used by the Nizari Ismaili community of South Asia for recording religious literature. It is one of two Landa scripts—the other being Gurmukhi—that were developed into formal liturgical scripts for use by religious communities. It is still used today.

Khudawadi is a Landa-based script that was used to write the Sindhi language spoken in India and Pakistan. It is related to Sharada. Known as the shopkeeper and merchant script, it was used for routine writing, accounting, and other commercial purposes.

The Multani script was used write the Seraiki language of eastern and southeastern Pakistan during the 19th and 20th centuries. Multani is related to Gurmukhi and more distantly related to Khudawadi and Khojki. It was used for routine writing and commercial activities.

Tirhuta, another Brahmi-based script, is related to the Bengali, Newari, and Oriya scripts. Tirhuta was the traditional writing system for the Maithili language, which is spoken by more than 35 million people in parts of India and Nepal. Maithili is an official regional language of India and the second most spoken language in Nepal.

Modi is another Brahmi-based script mainly used to write Marathi, a language spoken in western and central India. It emerged in the 16th century and derives from the Nagari scripts. It is still used some today.

Nandinagari is a Brahmi-based abugida that was used in southern India between the 11th and 19th centuries for manuscripts and inscriptions in Sanskrit. It is related to Devanagari. The script was also used for writing Kannada in Karnataka.

Grantha, a script with a long history, is used to write the Sanskrit language in parts of South India, Sri Lanka and elsewhere. It is in daily use by Vedic scholars and Hindu temple priests.

Tulu-Tigalari is a historic script attested in a large number of manuscripts from Karnataka and northern Kerala dating to as early as 1300 CE. It was used to write Sanskrit, Tulu, and Malayalam, but most attestations are manuscripts of Sanskrit reli-

gious texts written by Shivalli, Havyaka, and Kota brahmins. The script is known by a wide variety of names. It is currently undergoing revival among Tulu speakers in Karnataka, with some innovations, as a modern writing system alternative to the Kannada script for that language.

Dives Akuru is a Brahmi-derived script used to write the Dhivehi language on the Maldives from the 9th to the 20th centuries. The script is most closely related to a medieval form of the Sinhala script.

Ahom is a script of northeast India that dates to about the 16th century and was used primarily to write the Tai Ahom language. The script has seen a revival in the 20th century, and continues in some use today.

Sora Sompeng is used to write the Sora language spoken by the Sora people, who live in eastern India between the Oriya- and Telugu-speaking populations. The script was created in 1936 and is used in religious contexts.

15.1 Syloti Nagri

15.1.1 Syloti Nagri: U+A800–U+A82F

Syloti Nagri is a lesser-known Brahmi-derived script used for writing the Sylheti language. Sylheti is an Indo-European language spoken by some 5 million speakers in the Barak Valley region of northeast Bangladesh and southeast Assam in India. Worldwide there may be as many as 10 million speakers. Sylheti has commonly been regarded as a dialect of Bengali, with which it shares a high proportion of vocabulary.

The Syloti Nagri script has 27 consonant letters with an inherent vowel of /o/ and 5 independent vowel letters. There are 5 dependent vowel signs that are attached to a consonant letter. Unlike Devanagari, there are no vowel signs that appear to the left of their associated consonant.

Only two proper diacritics are encoded to support Syloti Nagri: *anusvara* and *hasanta*. Aside from its traditional Indic designation, *anusvara* can also be considered a final form for the sequence /-ng/, which does not have a base glyph in Syloti Nagri because it does not occur in other positions. *Anusvara* can also occur with the vowels U+A824 ◌ᠲ SYLOTI NAGRI VOWEL SIGN I and U+A826 ◌ SYLOTI NAGRI VOWEL SIGN E, creating a potential problem with the display of both items. It is recommended that *anusvara* always occur in sequence after any vowel signs, as a final character.

Virama and Conjuncts. Conjuncts are not always necessary in contexts involving a dead consonant, nor are they limited to sequences involving dead consonants. They can also represent a variety of vowel + consonant (VC) syllables, such as *ar*, *al*, *as*, *at*, *ir*, and *it*, as well as the CCV combinations typical of other Indic scripts. In practice, it is rare to overtly indicate a dead consonant with an explicit *hasanta*, and not always obligatory to use a conjunct.

U+A806 SYLOTI NAGRI SIGN HASANTA, whose glyph is shaped like a circumflex, was introduced into the script relatively recently and is used in limited contexts. The character appears overtly in pedagogical materials introducing readers to the script. More commonly, the *hasanta* is inserted between consonants to represent a conjunct. Occasionally, it indicates a word-final consonant whose vowel is silenced; however, the *hasanta* is generally not required in such cases. A second *hasanta*, U+A82C SYLOTI NAGRI SIGN ALTERNATE HASANTA, specifically indicates a word-final consonant. The glyph for the alternate *hasanta*, resembles U+09CD BENGALI SIGN VIRAMA and is used when the glyph for circumflex-shaped *hasanta* would overhang the following space. The alternate *hasanta* has very limited modern-day use.

Digits. There are no unique Syloti Nagri digits. When digits do appear in Syloti Nagri texts, they are generally Bengali forms. Any font designed to support Syloti Nagri

should include the Bengali digits because there is no guarantee that they would otherwise exist in a user's computing environment. They should use the corresponding Bengali block code points, U+09E6..U+09EF.

Punctuation. With the advent of digital type and the modernization of the Syloti Nagri script, one can expect to find all of the traditional punctuation marks borrowed from the Latin typography: *period*, *comma*, *colon*, *semicolon*, *question mark*, and so on. In addition, the Devanagari *single danda* and *double danda* are used with great frequency.

Poetry Marks. Four native poetry marks are included in the Syloti Nagri block. The script also makes use of U+2055 ⁑ FLOWER PUNCTUATION MARK (in the General Punctuation block) as a poetry mark.

15.2 Kaithi

15.2.1 Kaithi: U+11080–U+110CF

Kaithi, properly transliterated Kaithī, is a North Indian script, related to the Devanagari and Gujarati scripts. It was used in the area of the present-day states of Bihar and Uttar Pradesh in northern India.

Kaithi was employed for administrative purposes, commercial transactions, correspondence, and personal records, as well as to write religious and literary materials. As a means of administrative communication, the script was in use at least from the 16th century until the early 20th century, when it was eventually eclipsed by Devanagari. Kaithi was used to write Bhojpuri, Magahi, Awadhi, Maithili, Urdu, and other languages related to Hindi.

Standards. There is no preexisting character encoding standard for the Kaithi script. The repertoire encoded in this block is based on the standard form of Kaithi developed by the British government of Bihar and the British provinces of northwest India in the 19th century. A few additional Kaithi characters found in manuscripts, printed books, alphabet charts, and other inventories of the script are also included.

Styles. There are three presentation styles of the Kaithi script, each generally associated with a different language: Bhojpuri, Magahi, or Maithili. The Magahi style was adopted for official purposes in the state of Bihar, and is the basis for the representative glyphs in the code charts.

Rendering Behavior. Kaithi is a Brahmi-derived script closely related to Devanagari. In general, the rules for Devanagari rendering apply to Kaithi as well. For more information, see *Section 12.1, Devanagari*.

Vowel Letters. An independent Kaithi letter for *vocalic r* is represented by the consonant-vowel combination: U+110A9 KAITHI LETTER RA and U+110B2 KAITHI VOWEL SIGN II.

In print, the distinction between short and long forms of *i* and *u* is maintained. However, in handwritten text, there is a tendency to use the long vowels for both lengths.

Consonant Conjuncts. Consonant clusters were handled in various ways in Kaithi. Some spoken languages that used the Kaithi script simplified clusters by inserting a vowel between the consonants, or through metathesis. When no such simplification occurred, conjuncts were represented in different ways: by ligatures, as the combination of the half-form of the first consonant and the following consonant, with an explicit virama (U+110B9 KAITHI SIGN VIRAMA) between two consonants, or as two consonants without a virama.

Consonant conjuncts in Kaithi are represented with a virama between the two consonants in the conjunct. For example, the ordinary representation of the conjunct *mba* would be by the sequence:

U+110A7 KAITHI LETTER MA + U+110B9 KAITHI SIGN VIRAMA + U+110A5 KAITHI LETTER BA

Consonant conjuncts may be rendered in distinct ways. Where there is a need to render conjuncts in the exact form as they appear in a particular source document, U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER can be used to request the appropriate presentation by the rendering system. For example, to display the explicitly ligated glyph ⋈ for the conjunct *mba*, U+200D ZERO WIDTH JOINER is inserted after the virama:

U+110A7 KAITHI LETTER MA + U+110B9 KAITHI SIGN VIRAMA + U+200D ZERO WIDTH JOINER + U+110A5 KAITHI LETTER BA

To block use of a ligated glyph for the conjunct, and instead to display the conjunct with an explicit virama, U+200C ZERO WIDTH NON-JOINER is inserted after the virama:

U+110A7 KAITHI LETTER MA + U+110B9 KAITHI SIGN VIRAMA + U+200C ZERO WIDTH NON-JOINER + U+110A5 KAITHI LETTER BA

Conjuncts composed of a nasal and a consonant may be written either as a ligature with the half-form of the appropriate class nasal letter, or the full form of the nasal letter with an explicit virama (U+110B9 KAITHI SIGN VIRAMA) and consonant. In Grierson's *Linguistic Survey of India*, however, U+110A2 KAITHI LETTER NA is used for all articulation classes, both in ligatures and when the full form of the nasal appears with the virama.

Ruled Lines. Kaithi, unlike Devanagari, does not employ a headstroke. While several manuscripts and books show a headstroke similar to that of Devanagari, the line is actually a ruled line used for emphasis, titling or sectioning, and is not broken between individual letters. Some Kaithi fonts, however, were designed with a headstroke, but the line is not broken between individual letters, as would occur in Devanagari.

Nukta. Kaithi includes a nukta sign, U+110BA KAITHI SIGN NUKTA, a dot which is used as a diacritic below various consonants to form new letters. For example, the nukta is used to distinguish the sound *va* from *ba*. The precomposed character U+110AB KAITHI LETTER VA is separately encoded, and has a canonical decomposition into the sequence of U+110A5 KAITHI LETTER BA plus U+110BA KAITHI SIGN NUKTA. Precomposed characters are also encoded for two other Kaithi letters, *rha* and *dddha*.

The glyph for U+110A8 KAITHI LETTER YA may appear with or without a nukta. Because the form without the nukta is considered a glyph variant, it is not separately encoded as a character. The representative glyph used in the chart contains the dot. The nukta diacritic also marks letters representing some sounds in Urdu or sounds not native to Hindi. No precomposed characters are encoded in those cases, and such letters must be represented by a base character followed by the nukta.

Punctuation. A number of Kaithi-specific punctuation marks are encoded. Two marks designate the ends of text sections: U+110BE KAITHI SECTION MARK, which generally indicates the end of a sentence, and U+110BF KAITHI DOUBLE SECTION MARK, which delimits larger blocks of text, such as paragraphs. Both section marks are generally drawn so that their glyphs extend to the edge of the text margins, particularly in manuscripts.

The character U+110BD KAITHI NUMBER SIGN is a format control that interacts with digits. It occurs below a digit or sequence of digits, indicating a numerical reference. The related character U+110CD KAITHI NUMBER SIGN ABOVE occurs above a digit or sequence of digits, and indicates a number in an itemized list, similar to U+2116 NUMERO SIGN. Like U+0600 ARABIC NUMBER SIGN and the other Arabic signs that span numbers (see *Section 9.2, Arabic*), these Kaithi format controls precede the numbers they graphically interact with, rather than following them. U+110BC KAITHI ENUMERATION SIGN is a standalone, spacing symbol for inline usage.

U+110BB KAITHI ABBREVIATION SIGN, shaped like a small circle, is used in Kaithi to indicate abbreviations. This mark is placed at the point of elision or after a ligature to indicate common words or phrases that are abbreviated, in a similar way to U+0970 DEVANAGARI ABBREVIATION SIGN.

Kaithi makes use of two script-specific dandas: U+110C0 KAITHI DANDA and U+110C1 KAITHI DOUBLE DANDA.

For other punctuation marks occurring in Kaithi texts, available Unicode characters may be used. A cross-shaped character, used to mark phrase boundaries, can be represented by U+002B PLUS SIGN. For hyphenation, users should follow whatever is the recommended practice found in similar Indic script traditions, which might be U+2010 HYPHEN or U+002D HYPHEN-MINUS. For dot-like marks that appear as word-separators, U+2E31 WORD SEPARATOR MIDDLE DOT, or, if the word boundary is more like a dash, U+2010 HYPHEN can be used.

Digits. The digits in Kaithi are considered to be stylistic variants of those used in Devanagari. Hence the Devanagari digits located at U+0966..U+096F should be employed. To indicate fractions and unit marks, Kaithi uses characters encoded in the Common Indic Number Forms block, U+A830..U+A839.

15.3 Sharada

15.3.1 Sharada: U+11180–U+111DF

Sharada is a historical script that was used to write Sanskrit, Kashmiri, and other languages of northern South Asia. It served as the principal inscriptional and literary script of Kashmir from the 8th century CE until the 20th century. In the 19th century, expanded use of the Arabic script to write Kashmiri and the growth of Devanagari contributed to the marginalization of Sharada. Today the script is employed in a limited capacity by Kashmiri pandits for horoscopes and ritual purposes.

Rendering Behavior. Sharada is a Brahmi-based script, closely related to Devanagari. In general, the rules for Devanagari rendering apply to Sharada as well. For more information, see *Section 12.1, Devanagari*.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 15-1* shows the Sharada letter that can be analyzed, the single code point that should be used to represent it in text, and the sequence of code points resulting from analysis that should not be used. In contrast, the atomic U+111C4 SHARADA OM is not recommended for use; the *om* should be written in Sharada with a character sequence, instead.

Table 15-1. Sharada Vowel Letters and *om*

For	Use	Do Not Use
॒	1118E ॒	<1118D ॒, 111BC ॒>
॒	<1118F ॒, 11180 ॒>	111C4 ॒

Ruled Lines. While the headstroke is an important structural feature of a character's glyph in Sharada, there is no rule governing the joining of headstrokes of characters to other characters. The variation was probably due to scribal preference, and should be handled at the font level.

Virama. The U+111C0 ॒ SHARADA SIGN VIRAMA is a spacing mark, written to the right of the consonant letter it modifies. Semantically, it is identical to the Devanagari *virama* and other similar Indic scripts.

Candrabindu and Avagraha. U+11180 ॒ SHARADA SIGN CANDRABINDU indicates nasalization of a vowel. It may appear in manuscripts in an inverted form but with no semantic difference. Such glyph variants should be handled in the font. U+111C1 ॒ SHARADA SIGN AVAGRAHA represents the elision of a word-initial *a*. Unlike the usual practice in Devanagari in which the *avagraha* is written at the normal letter height and attaches to the top stroke of the following character, the *avagraha* in Sharada is written at or below the baseline and does not connect to the neighboring letter.

Jihvamuliya and Upadhmaniya. The velar and labial allophones of /h/, followed by voiceless velar and labial stops respectively, are written in Sharada with separate signs, U+111C2 [ꠂ] SHARADA SIGN JIHVAMULIYA and U+111C3 [ꠃ] SHARADA SIGN UPADHMANIYA. These two signs have the properties of a letter and appear only in stacked conjuncts without the use of *virama*. *Jihvamuliya* is used to represent the velar fricative [x] in the context of a following voiceless velar stop:

U+111C2 [ꠂ] *jihvamuliya* + U+11191 𑆑 *ka* → 𑆑

U+111C2 [ꠂ] *jihvamuliya* + U+11192 𑆒 *kha* → 𑆒

Upadhmaniya is used to represent the bilabial fricative [ɸ] in the context of a following voiceless labial stop:

U+111C3 [ꠃ] *upadhmaniya* + U+111A5 𑆥 *pa* → 𑆥

U+111C3 [ꠃ] *upadhmaniya* + U+111A6 𑆦 *pha* → 𑆦

Punctuation. U+111C7 · SHARADA ABBREVIATION SIGN appears after letters or combinations of letters. It marks the sequence as an abbreviation. A word separator, U+111C8 ⸰ SHARADA SEPARATOR, indicates word and other boundaries. Sharada also makes use of two script-specific dandas: U+111C5 । SHARADA DANDA and U+111C6 ॥ SHARADA DOUBLE DANDA.

Digits. Sharada has a distinctive set of digits encoded in the range U+111D0..U+111D9.

15.4 Takri

15.4.1 Takri: U+11680–U+116CF

Takri is a script used in northern India and surrounding countries in South Asia, including the areas that comprise present-day Jammu and Kashmir, Himachal Pradesh, Punjab, and Uttarakhand. It is the traditional writing system for the Chambeali and Dogri languages, as well as several "Pahari" languages, such as Jaunsari, Kulvi, and Mandeali. It is related to the Gurmukhi, Landa, and Sharada scripts. Like other Brahmi-derived scripts, Takri is an *abugida*, with consonants taking an inherent vowel unless accompanied by a vowel marker or the *virama* (vowel killer).

Takri is descended from Sharada through an intermediate form known as Devāśeṣa, which emerged in the 14th century. Devāśeṣa was a script used for religious and official purposes, while its popular form, known as Takri, was used for commercial and informal purposes. Takri became differentiated from Devāśeṣa during the 16th century. In its various regional manifestations, Takri served as the official script of several princely states of northern and northwestern India from the 17th century until the middle of the 20th century. Until the late 19th century, Takri was used concurrently with Devanagari, but it was gradually replaced by the latter.

Owing to its use as both an official and a popular script, Takri appears in numerous records, from manuscripts to inscriptions to postage stamps. There are efforts to revive the use of Takri for languages such as Dogri, Kishtwari, and Kulvi as a means of preserving access to these language's literatures.

There is no universal, standard form of Takri. Where Takri was standardized, the reformed script was limited to a particular polity, such as a kingdom or a princely state. The representative glyphs shown in the code charts are taken mainly from the forms used in a variant established as the official script for writing the Chambeali language in the former Chamba State, now in Himachal Pradesh, India. There are a number of other regional varieties of Takri that have varying letterforms, sometimes quite different from the representative forms shown in the code charts. Such regional forms are considered glyphic variants and should be handled at the font level.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. _Table 15-2_ shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 15-2. Takri Vowel Letters

For	Use	Do Not Use
𑚁	11681	<11680, 116AD>

ẽ	11687	<11686, 116B2>
ẵ	11688	<11680, 116B4>
ẫ	11689	<11680, 116B5>

Consonant Conjuncts. Conjuncts in Takri are infrequent and, when written, consist of two consonants, the second of which is always *ya*, *ra*, or *ha*. Takri *ya* is written as a subjoining form; Takri *ra* can be written as a ligature or a subjoining form; and Takri *ha* is written as a half-form.

Nukta. A combining *nukta* character is encoded as U+116B7 TAKRI SIGN NUKTA. Characters that use this sound, mainly loan words and words from other languages, may be represented using the base character plus *nukta*.

Headlines. Unlike Devanagari, headlines are not generally used in Takri. However, headlines do appear in the glyph shapes of certain Takri letters. The headline is an intrinsic feature of glyph shapes in some regional varieties such as Dogra Akkhar, where it appears to be inspired by the design of Devanagari characters. There are no fixed rules for the joining of headlines. For example, the headlines of two sequential characters possessing headlines are left unjoined in Chambeali, while the headlines of a letter and a vowel sign are joined in printed Dogra Akkhar.

Punctuation. Takri uses U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA from Devanagari.

Fractions. Fraction signs and currency marks found in Takri documents use the characters in the Common Indic Number Forms block (U+A830..U+A83F).

15.5 Siddham

15.5.1 Siddham: U+11580–U+115FF

Siddham is a Brahmi-based writing system that originated in India, and is presently used primarily in East Asia. The script is also known as Siddhamātṛkā and Kuṭila. The name Siddhamatrika has broad historic and regional usage throughout India and East Asia. However, modern usage is most strongly associated with the Shingon and Tendai Buddhist traditions in Japan, where the script is also known as Bonji. The representative glyphs in the code charts are based upon Japanese forms of Siddham characters.

The historical record shows the use of Siddham in Central Asia, but the predominant examples are of its use for writing Sanskrit in China, Japan, and Korea, notably for Buddhist manuscripts. Today, it is mainly used for ceremonial and ritualistic purposes associated with esoteric Buddhist practices.

Siddham is most closely related to Sharada, another Brahmi-based script that originated in Kashmir.

Nukta. The sign U+115C0 ◌ SIDDHAM SIGN NUKTA is used for transcribing sounds that are not native to the writing system. The *nukta sign* is not a traditional Siddham character, but it is part of modern Siddham, so that it can accommodate the writing of Japanese and English.

Vowels. The Siddham vowel signs for *u* and *uu* may appear in two forms. The regular forms, called "cloud" forms, are represented by U+115B2 SIDDHAM VOWEL SIGN U and U+115B3 SIDDHAM VOWEL SIGN UU. Alternate vowel sign forms, referred to as "warbler" forms, are represented instead by U+115DC SIDDHAM VOWEL SIGN ALTERNATE U and U+115DD SIDDHAM VOWEL SIGN ALTERNATE UU.

The combination of *ra* and *u* should be written with the sequence <U+115A8 𑖨 SIDDHAM LETTER RA, U+115DC ◌ SIDDHAM VOWEL SIGN ALTERNATE U> and rendered as 𑖨. For the combination *ra* and *uu*, the form 𑖨 should be employed, represented by the sequence <U+115A8 SIDDHAM LETTER RA, U+115DD SIDDHAM VOWEL SIGN ALTERNATE UU>.

Virama and Conjuncts. The virama, U+115BF ◌ SIDDHAM SIGN VIRAMA, is identical to the corresponding character in Devanagari and silences the inherent vowel of a consonant. The default rendering of the Siddham *virama* is as a visible sign.

Consonant clusters in Siddham are written as conjuncts and follow the same model as conjuncts in Devanagari. Conjuncts are represented using the Siddham *virama*, which is written between each consonant in the cluster. Conjuncts may be written

vertically, horizontally, or as independent ligatures. There are traditional Chinese and Japanese tabulations for Siddham conjuncts.

Siddham conjuncts may represent clusters with a large number of consonants. For example, *rkṣvrya* is a conjunct cluster produced by a sequence of six conjuncts, as shown in *Figure 15-1*.

Figure 15-1. Siddham Consonant Cluster

Head Marks. The mark U+115C1 ᴏ SIDDHAM SIGN SIDDHAM is written at the beginning of a text. Paleographically, the sign corresponds to characters used in other scripts, such as U+0FD3 ᴏ TIBETAN MARK INITIAL BRDA RNYING YIG MGO MDUN MA. It represents the Sanskrit word *siddham*, "accomplished," and the phrase *siddhi-rastu*, "may there be success." A vertically-oriented glyph variant is used for vertical text layout.

Repetition Marks. Three marks, U+115C6 ᴣ SIDDHAM REPETITION MARK-1, U+115C7 ᴐ SIDDHAM REPETITION MARK-2, and U+115C8 ᴢ SIDDHAM REPETITION MARK-3 are used to indicate the text repetition. They are written after the text that is to be repeated.

Section Signs. A set of fourteen section marks are used in Siddham to indicate the ends of sentences, phrases, verses, and sections. They appear in manuscripts and script manuals. According to the Shingon philosophy, the characters possess esoteric qualities that relay information regarding the interpretation of the text.

Punctuation. There are five other punctuation marks encoded for Siddham, as shown in *Table 15-3*. Both Siddham *danda* and Siddham *double danda* have graphical variants used in informal Japanese writing of Siddham.

Table 15-3. Siddham Punctuation Characters

Code Point and Name		Purpose
115C2	ᴣ SIDDHAM DANDA	marks the end of sentences and other short text sections
115C3	ᴫ SIDDHAM DOUBLE DANDA	used at the end of paragraphs and larger text blocks
115C4	⋅ SIDDHAM SEPARATOR DOT	marks boundaries between syllables, words, and phrases; written at the head-height.
115C5	ǀ SIDDHAM SEPARATOR BAR	marks boundaries between syllables, words, and phrases
115C9	⫴ SIDDHAM END OF TEXT MARK	indicates the end or completion of a text

15.6 Mahajani

15.6.1 Mahajani: U+11150–U+1117F

Mahajani is a Brahmi-based writing system that was commonly used across northern India until the middle of the 20th century. It is a specialized commercial script used for writing accounts and financial records. It was used for recording several languages: Hindi, Marwari, and Punjabi. Mahajani was taught and used as a medium of education in Punjab, Rajasthan, Uttar Pradesh, Bihar, and Madhya Pradesh in schools where students from merchant and trading communities learned the script and other writing skills required for business. The name "Mahajani" refers to bankers and money lenders, who were the primary users of the script. The majority of Mahajani records are account books. Although the Mahajani script is no longer in general use, it is an important key to the historical financial records of northern India.

Mahajani has similarities to Landa, Kaithi, and Devanagari. In structure and orthography, Mahajani resembles scripts of the Landa family used in Punjab and Sindh, which are related to Sharada.

Structure. Mahajani is written from left to right. It is based upon the Brahmi model, but it is structurally simpler and behaves as an alphabet. Vowel signs are not used, and there is no virama. Consonant clusters are not written in Mahajani using half-forms or ligatures (except for one ligature for *shri*), or even a visible virama. The elements of a consonant cluster are written sequentially using regular consonant letters.

Vowel signs are not written. Consonant letters theoretically bear the inherent vowel /a/, but the glyph for *ka* for example represents not only *ka*, but also any one of the syllables *ka*, *kā*, *ki*, *kī*, *ke*, and so on. In cases where greater precision is required, a vowel letter may be written after a consonant to convey the intended vocalic context. In general, the value of a consonant letter must be inferred at the morphological level.

Nasalization is not represented using special signs, such as *anusvara*. Instead U+11167 MAHAJANI LETTER NA is used in cases where nasalization is explicitly recorded. In several cases, words are written simply with nasalization deleted.

U+11173 MAHAJANI SIGN NUKTA is used for writing sounds that are not represented by a unique character, such as allophonic variants and sounds that occur in local dialects or in loanwords. It has limited use in Mahajani.

Several letters have glyphic variants. Those variants are not separately encoded.

Digits. Mahajani does not have distinctive script-specific digits. The Devanagari digits located at U+0966..U+096F should be used.

Other Symbols. Fraction signs and unit marks are found in Mahajani documents, and may be represented using the characters encoded in the "Common Indic Number Forms" block.

Punctuation. Mahajani employs a dash, middle dot, and colon, which should be represented by the corresponding Latin characters. For the *dandas*, Mahajani employs U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA. Mahajani also contains two other script-specific punctuation signs, U+11174 MAHA-JANI ABBREVIATION SIGN and U+11175 MAHAJANI SECTION MARK. There are no formal rules for punctuation, and word spacing is not generally observed.

15.7 Khojki

15.7.1 Khojki: U+11200–U+1124F

Khojki is a writing system used by the Nizari Ismaili community of South Asia for recording religious literature. It was developed in Sindh, now in Pakistan, for representing the Sindhi language. The script spread to surrounding regions and was used for writing Gujarati, Punjabi, and Siraiki, as well as several languages related to Hindi. It was also used for writing Arabic and Persian. Popular Nizari Ismaili tradition states that Khojki was invented and propagated by Pir Sadruddin, an Ismaili missionary.

Khojki is one of two Landa scripts that were developed into formal liturgical scripts for use by religious communities; the other is Gurmukhi, which was developed for writing the sacred literature of the Sikh tradition.

Khojki is also called "Sindhi" and "Khwajah Sindhi." Khojki was in use by the 16th century CE, as attested by manuscript evidence. The printing of Khojki books flourished after Laljibhai Devraj produced metal types for Khojki in Germany for use at his Khoja Sindhi Printing Press in Mumbai.

While usage of Khojki has declined over the past century, it is used wherever Nizari Ismaili Muslims of South Asian origin reside. The largest communities are found in Pakistan, India, Canada, United States, the United Kingdom, Kenya, Tanzania, and Uganda. Khojki primers continue to be published in Pakistan for teaching the script. Khojki manuscripts and books are used in Ismaili ceremonies not only in South Asia, but in east and south Africa, where large diaspora communities formed by the 19th century. The script was also used by communities related to the Nizari Ismailis, such as the Imamshahis of Gujarat.

Structure. The general structure of Khojki is similar to that of other Brahmi-derived Indic scripts. It is written from left to right.

Khojki has a smaller repertoire of independent vowel letters than other Brahmi-derived scripts. Conventionally, the letters U+11202 KHOJKI LETTER I and U+11203 KHOJKI LETTER U are used for writing both short and long forms of *i* and *u*, respectively. However, some Khojki texts distinguish between the short and long forms of *i*. Those texts should use U+11202 KHOJKI LETTER I to represent *long i* and U+11240 KHOJKI LETTER SHORT I to represent *short i*. The letters U+11205 KHOJKI LETTER AI and U+11207 KHOJKI LETTER AU represent diphthongs. Although they are attested in manuscripts and books, Khojki originally did not have unique letters for these vowels. In early Khojki records, diphthongs are generally represented as digraphs. Several variant forms of vowel letters are also attested.

The repertoire of dependent vowel signs is larger than that of independent vowel letters. There are separate signs for U+1122D KHOJKI VOWEL SIGN I and U+1122E KHOJKI VOWEL SIGN II, but no form for *uu*. Instead, the single sign U+1122F KHOJKI VOWEL SIGN U is used for both short and long forms. U+11232 KHOJKI VOWEL SIGN O is often written by placing the U+11230 KHOJKI VOWEL SIGN E element above the consonant letter.

Geminate consonants are marked by the U+11237 KHOJKI SIGN SHADDA, written above the consonant letter that is doubled. The positioning may change in relation to vowel signs.

Nasalization is indicated by the sign U+11234 KHOJKI SIGN ANUSVARA. It is written to the right of the letter or sign with which it combines.

U+11235 KHOJKI SIGN VIRAMA is identical in function to corresponding characters in other Indic scripts. It is written to the right of a consonant letter.

U+11236 KHOJKI SIGN NUKTA is used for producing characters to represent sounds not native to Sindhi. The sign may be written with vowel letters, vowel signs, and consonant letters. The *nukta* is written above a letter.

Vowels. Khojki vowel letters and vowel signs are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 15-4* shows the letters and signs that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 15-4. Khojki Vowels

For	Use	Do Not Use
ꓲꓲ	11201	<11200, 1122C>
ꓲꓲ	11202	<11240, 1122E>
ꓲ	11203	<11206, 1122C>
ꓲꓲ	11205	<11200, 11231>
ꓲꓲ	11207	<11200, 11233> or <11200, 1122C, 11231>
ꓲ	11232	<1122C, 11230>
ꓲ	11233	<1122C, 11231>

Punctuation. Khojki separates words using U+1123A KHOJKI WORD SEPARATOR. U+11238 KHOJKI DANDA and U+11239 KHOJKI DOUBLE DANDA are used to mark the end of sentences. The DOUBLE DANDA is also used to mark verse sections. Typically, DOUBLE DANDA is written with U+1123A KHOJKI WORD SEPARATOR to the left and right of verse numbers.

Section marks appear frequently in Khojki manuscripts as punctuation that delimits the end of a section or another larger block of text. The U+1123B KHOJKI SECTION MARK is generally used to mark the end of a sentence, while U+1123C KHOJKI DOUBLE SECTION MARK is used to delimit larger blocks of text, such as paragraphs. Both generally extend to the margin of the text-block.

Latin punctuation marks are also used in printed Khojki.

U+1123D KHOJKI ABBREVIATION SIGN is used for marking abbreviations.

Digits. Khojki makes use of Gujarati digits U+0AE6 through U+0AEF.

15.8 Dogra

15.8.1 Dogra: U+11800–U+1184F

In the 17th century, the Dogra script was used to write the Dogri language in Jammu and Kashmir in the northern region of the Indian subcontinent. Dogri is an Indo-Aryan language now usually written with the Devanagari script. The Dogra script was standardized in the 1860s, and is closely related to the Takri script. The official form, known as "Name Dogra Akkar" or "New Dogra Script," appears in administrative documents, on currency, postcards, postage stamps, and in literary works. The unofficial, common written form of the script is called "Old Dogra." The glyphs in the code chart are based on New Dogra.

Structure. Dogra is an *abugida*, based on Brahmi. It is written left to right. The script includes a *virama*, U+11839 DOGRA SIGN VIRAMA, to create conjuncts and to suppress the inherent vowel.

Vowels. Because the glyphs for Dogra vowel letters changed over time, the phonetic value of three vowel letters varies between New and Old Dogra. Old Dogra uses U+11802 DOGRA LETTER I for *u*, U+11803 DOGRA LETTER II for *i*, and U+11804 DOGRA LETTER U for *o* and *au*. The shapes of the vowel signs also vary between Old and New Dogra. Distinct fonts can be used to reflect the Old Dogra vowel shapes, as opposed to the New Dogra shapes.

A feature of Dogra is that the dependent vowel may be represented either by the independent vowel letter, or by the dependent vowel sign. For example, the syllable *ke* may be represented by ਰ𑄘 <*ka, e*> or ਰ̄ <*ka, vowel sign e*>.

Characters Used to Represent Sanskrit. U+11831 DOGRA VOWEL SIGN VOCALIC R, U+11832 DOGRA VOWEL SIGN VOCALIC RR, and U+11828 DOGRA LETTER SSA are used in New Dogra to represent sounds of Sanskrit origin.

Consonant Conjuncts. Consonant clusters in Dogra may be rendered in different ways. The most common method is to place a *virama* beneath each bare consonant. Certain consonant clusters may also be written as conjuncts. A conjunct may be an atomic ligature, such as ੬ *kṣa* (represented with <*ka, virama, ssa*>), or a looser ligature, such as ੯ *ṣṭa* (<*sa, virama, tta*>), in which the individual shapes of each letter are visible.

In particular, although Dogra does not normally use *repha* to represent the initial *ra* in a consonant cluster, a non-initial *ra* is sometimes conjoined to form a ligature. A conjoined non-initial *ra* is usually attached below the base letter, in a somewhat reduced form. Depending on the graphical structure of the preceding consonant, the non-initial *ra* may also appear to be the base of the cluster, with the preceding conso-

nant taking a half-form instead. For example, New Dogra consistently uses the conjunct 𑇩 *śra* (*<sha*, *virama*, *ra>*) in the Sanskrit honorific *śrī*, which shows a half-form of *śa*.

Other Symbols. U+11837 DOGRA SIGN ANUSVARA indicates nasalization, and U+11838 DOGRA SIGN VISARGA indicates post-vocalic aspiration in words of Sanskrit origin, while U+1183A DOGRA SIGN NUKTA is used to transcribe sounds that are not native to the Dogri language.

Punctuation. U+1183B DOGRA ABBREVIATION SIGN denotes abbreviations. U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA indicate the ends of sentences and paragraphs.

Digits and Number Forms. Digits in Dogra vary across written and printed sources: some Old Dogra digits resemble Takri digits, while digits in some New Dogra documents resemble Devanagari. Because of this wide variation, script-specific digits have not been encoded. Devanagari digits should be used to represent digits in Dogra text. For representation of Dogra fraction and currency signs, use characters from the Common Indic Number Forms block.

15.9 Khudawadi

15.9.1 Khudawadi: U+112B0–U+112FF

Khudawadi is a script used historically for writing the Sindhi language, which is spoken in India and Pakistan. Official forms of Khudawadi are known as "Hindi Sindhi," "Hindu Sindhi," and "Standard Sindhi." Khudawadi is a Landa-based script and related to Sharada. Like other Landa writing systems, Khudawadi is a mercantile script used for routine writing, accounting, and other commercial purposes and was known as the shopkeeper and merchant script. It is associated with the merchant communities of Hyderabad, Sindh. In addition to mercantile records, Khudawadi was used in education, book printing, and for court records.

In the 1860s, Khudawadi was chosen as the basis for a written standard for education and administration in Sindh and was developed as an official language. Official Khudawadi possesses unique characters for each vowel and consonant sound of the Sindhi language, as well as vowel signs. In the late 19th century, an Arabic-based script became the official writing system for Sindhi in Pakistan and India. Sindhi is also written in the Devanagari script in India. Khudawadi is now obsolete.

Structure. The general structure of Khudawadi is similar to that of other Brahmi-based Indic scripts. It is written from left to right.

Vowel Letters. Some independent vowel letters may be represented using a combination of a base vowel letter and a dependent vowel sign. This practice is not recommended. The atomic character for the independent vowel letter should always be used.

Table 15-5. Khudawadi Vowel Letters

For	Use	Do Not Use
ოι	112B1	112B0 + 112E0
ῶ	112B6	112B0 + 112E5
ῶ̇	112B7	112B0 + 112E6
ῶ̣	112B8	112B0 + 112E7
ῶ̤	112B9	112B0 + 112E8

Consonant Conjuncts. Consonant clusters generally consist of two consonants. These are written using a visible *virama*. The encoded representation is <C1 + *virama* + C2>. Half-forms and ligated conjunct forms are not attested.

Nasalization. U+112DF ◌̇ KHUDAWADI SIGN ANUSVARA is used for indicating nasalization.

Nukta. U+112E9 ◌ KHUDAWADI SIGN NUKTA is used for representing sounds not native to Sindhi, such as those that may occur in Persian and Arabic loanwords. Attested Khudawadi letters with *nukta* are shown in <u>Table 15-6</u>, along with the Arabic letters for which they substitute. JA + NUKTA, pronounced *za*, corresponds to a number of distinct Arabic letters.

Table 15-6. Representation of Arabic Sounds in Khudawadi

Sound	Khudawadi		Arabic
kha	�	KHA + NUKTA	U+062E ARABIC LETTER KHAH
ġa	𑌂	GA + NUKTA	U+063A ARABIC LETTER GHAIN
za	�꣢	JA + NUKTA	U+0630 ARABIC LETTER THAL U+0632 ARABIC LETTER ZAIN U+0636 ARABIC LETTER DAD U+0638 ARABIC LETTER ZAH
fa	𑌽	PHA + NUKTA	U+0641 ARABIC LETTER FEH

In principle, the *nukta* may be written with any Khudawadi vowel or consonant letter. If other combining marks, such as a dependent vowel sign or *anusvara*, also occur in a combining sequence applied to that base character, then the convention is to represent the *nukta* first in the combining sequence.

Punctuation. The Khudawadi uses *dandas* and European punctuation, such as periods, dashes, colons, and semi-colons. Khudawadi *dandas* are unified with those of Devanagari. Line breaking for Khudawadi characters follows the rules for Devanagari.

Digits. Khudawadi has a full set of decimal digits. Fraction signs and currency marks are attested in Khudawadi records. These may be represented using characters in the Common Indic Number Forms block found at U+A830..U+A83F.

15.10 Multani

15.10.1 Multani: U+11280–U+112AF

The Multani script was used to write the Seraiki language, an Indo-Aryan language spoken in the Punjab in eastern Pakistan and the northern Sindh area of southeastern Pakistan. Multani is a Landa-based script, related to Gurmukhi, and distantly related to Khudawadi and Khojki. The script, also known as Karikki or Sarai, was used for routine writing and commercial activities. The first book in the Multani script was published in 1819. By the latter half of the 19th century, the British administration introduced the Arabic script as the standard for writing the languages of the Sindh, which led to the demise of various non-Arabic scripts, including Multani. The script continued to be used into the 20th century. Today Seraiki is written in the Arabic script.

There is no standard form of the Multani script. The representative glyphs shown in the code charts are based on printed forms from an 1819 version of the New Testament, with additional characters that are found only in handwritten documents. Such variant forms are considered glyphic variants and should be handled at the font level.

The script underwent orthographic changes in the first quarter of the 20th century, with a reduction in the character repertoire. The repertoire encoded in this block is based on the set of all characters that are distinctly attested.

Structure. Although Multani is based on the Brahmi model, it is closer in structure to an abjad than an *abugida*. There are four independent vowel letters, *a*, *i*, *u* and *e*, and no dependent vowel signs. Consonants theoretically possess the inherent /a/ vowel, but as vowels are not marked, the actual syllabic vowel of a consonant in running text is ambiguous and must be inferred from context. Consonant clusters are written using independent letters, rather than with conjuncts. There is no *virama*. Vowels are generally not written unless they occur in isolation, in word initial position, or in the final position of monosyllabic words.

The letter ᵥᵥ *a* is used to represent /a/, /a:/ and in some sources /e/ and /æ/. The letter ᵹ *i* represents /i/ and /i:/ and commonly the semivowel /j/. The letter ᵹ *u* represents /u/, /u:/ and /o/. The letter ᵔ *e* represents /e/, and in some sources /æ/ and /o/.

Digits. The Gurmukhi digits U+0A66..U+0A6F should be employed to represent digits in Multani.

Punctuation. Multani has only one script-specific punctuation mark, U+112A9 MULTANI SECTION MARK, which indicates the end of a sentence.

15.11 Tirhuta

15.11.1 Tirhuta: U+11480–U+114DF

Tirhuta was the traditional writing system for the Maithili language, which is spoken by more than 35 million people in the state of Bihar in India, and in the Koshi and Madhesh provinces of Nepal. Maithili is an official regional language of India and the second most spoken language in Nepal. Tirhuta is a Brahmi-based script derived from Gauḍī, or "Proto-Bengali," which evolved from the Kuṭila branch of Brahmi by the 10th century. It is related to the Bengali, Newari, and Oriya scripts, which are also descended from Gauḍī, and became differentiated from them by the 14th century.

Tirhuta remained the primary writing system for Maithili until the late 20th century, when it was replaced by Devanagari. The Tirhuta script forms the basis of scholarly and religious scribal traditions that have been associated with the Maithili and Sanskrit languages since the 14th century. Tirhuta continues to be used for writing manuscripts of religious and literary texts, as well as personal correspondence. Since the 1950s, various literary societies, such as the Maithili Akademi and Chetna Samiti, have been publishing literary, educational, and linguistic materials in Tirhuta. The script is also used in signage in Darbhanga and other districts of north Bihar, and as an optional script for writing the civil services examination in Bihar.

Although several Tirhuta characters, ligatures or combined shapes bear resemblance to those of Bengali, these similarities are superficial.

Structure. The general structure (phonetic order, matra reordering, use of *virama*, and so on) of Tirhuta is similar to that of other Brahmi-based Indic scripts. The script is written from left to right.

Vowels. Tirhuta uses independent vowel letters and corresponding combining vowel signs. The signs U+114BA ᴛɪʀʜᴜᴛᴀ ᴠᴏᴡᴇʟ ꜱɪɢɴ ꜱʜᴏʀᴛ ᴇ and U+114BD ᴛɪʀʜᴜᴛᴀ ᴠᴏᴡᴇʟ ꜱɪɢɴ ꜱʜᴏʀᴛ ᴏ do not have corresponding independent forms, because the sounds they represent do not occur in word initial position.

Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 15-7* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 15-7. Tirhuta Vowel Letters

For	Use	Do Not Use
था	11482	<11481, 114B0>
नृ	11489	<114AA, 114B5>

ৃ	1148A	<114AA, 114B6>
ৈ	1148C	<1148B, 114BA>
ৌ	1148E	<1148D, 114BA>

Consonants. Some of the 33 consonants look like Bengali consonants, but represent different sounds. For example, U+114A9 ᴛɪʀʜᴜᴛᴀ ʟᴇᴛᴛᴇʀ ʀᴀ has the same form as U+09AC ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ʙᴀ, and U+114AB ᴛɪʀʜᴜᴛᴀ ʟᴇᴛᴛᴇʀ ᴠᴀ has the same shape as U+09B0 ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ʀᴀ.

Consonants combined with vowel signs, combined in conjuncts, or appearing at the end of a word commonly use context-dependent ligatures or glyph combinations. These shapes also contrast with usage in Bengali. For example, the consonant-vowel combination <U+1149E ᴛɪʀʜᴜᴛᴀ ʟᴇᴛᴛᴇʀ ᴛᴀ, U+114B3 ᴛɪʀʜᴜᴛᴀ ᴠᴏᴡᴇʟ sɪɢɴ ᴜ> in Tirhuta produces the same shape as the conjunct <U+09A4 ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ᴛᴀ, U+09CD ʙᴇɴɢᴀʟɪ sɪɢɴ ᴠɪʀᴀᴍᴀ, U+09A4 ʙᴇɴɢᴀʟɪ ʟᴇᴛᴛᴇʀ ᴛᴀ> in the Bengali script.

All variant forms for letters, character elements and conjuncts in Tirhuta should be managed at the font level.

Virama. U+114C2 ᴛɪʀʜᴜᴛᴀ sɪɢɴ ᴠɪʀᴀᴍᴀ is identical in function to the corresponding character in other Indic scripts.

Nasalization. Nasalization is indicated by U+114BF ᴛɪʀʜᴜᴛᴀ sɪɢɴ ᴄᴀɴᴅʀᴀʙɪɴᴅᴜ and U+114C0 ᴛɪʀʜᴜᴛᴀ sɪɢɴ ᴀɴᴜsᴠᴀʀᴀ. These signs are written centered above the base. If written with an above-base sign or a letter with a graphical element that extends past the headstroke, they are placed to the right of such signs and elements.

Characters for Representing Sanskrit. Two characters are attested in Vedic and classical Sanskrit manuscripts written in Tirhuta. U+114C1 ᴛɪʀʜᴜᴛᴀ sɪɢɴ ᴠɪsᴀʀɢᴀ represents an allophone of *ra* or *sa* at word-final position in Sanskrit orthography. U+114C5 ᴛɪʀʜᴜᴛᴀ ɢᴠᴀɴɢ represents nasalization. It belongs to the same class of characters as U+1CE9 ᴠᴇᴅɪᴄ sɪɢɴ ᴀɴᴜsᴠᴀʀᴀ ᴀɴᴛᴀʀɢᴏᴍᴜᴋʜᴀ, U+1CEA ᴠᴇᴅɪᴄ sɪɢɴ ᴀɴᴜsᴠᴀʀᴀ ʙᴀʜɪʀɢᴏᴍᴜᴋʜᴀ, and so on.

Tirhuta also uses U+1CF2 ᴠᴇᴅɪᴄ sɪɢɴ ᴀʀᴅʜᴀᴠɪsᴀʀɢᴀ which can be found in the Vedic Extensions block.

Nukta. U+114C3 ᴛɪʀʜᴜᴛᴀ sɪɢɴ ɴᴜᴋᴛᴀ is used for writing sounds that are not represented by a unique character, such as allophonic variants and sounds that occur in local dialects or in loanwords. The *nukta* may be written with any vowel or consonant letter. If other combining marks, such as a vowel sign or *anusvara*, also appear with the base character, then the *nukta* is written first.

U+114A5 TIRHUTA LETTER BA and U+114AB TIRHUTA LETTER VA have shapes that include a dot, but this is not semantically equivalent to a *nukta*. These letters do not decompose to *nukta*, and are treated as atomic characters.

Punctuation. Tirhuta uses U+0964 DEVANAGARI DANDA and U+0965 DEVANAGARI DOUBLE DANDA from the Devanagari block.

Special Signs. U+114C6 TIRHUTA ABBREVIATION SIGN denotes abbreviations. There are also two special script-specific signs in Tirhuta. The first, U+11480 TIRHUTA ANJI, is used in the invocations of letters, manuscripts, books, and charts of the script. The sign *anji* is said to represent the tusk of the deity Ganesa, patron of learning. The second, U+114C7 TIRHUTA OM, contrasts with the Bengali sign for *om*, the latter being a simple combination of U+0993 BENGALI LETTER O plus U+0981 BENGALI SIGN CANDRABINDU.

Digits. Tirhuta has a full set of decimal digits.

Fractions. Number forms and unit marks are also found in Tirhuta documents. The most common of these are signs for writing fractions and currency, and they are represented using characters in the Common Indic Number Forms block (U+A830..U+A83F). They include U+A831 NORTH INDIC FRACTION ONE HALF, U+A832 NORTH INDIC FRACTION THREE QUARTERS, and so on, as well as U+A838 NORTH INDIC RUPEE MARK. Tirhuta also uses Bengali "currency numerators," such as U+09F4 BENGALI CURRENCY NUMERATOR ONE.

15.12 Modi

15.12.1 Modi: U+11600–U+1165F

Modi is a Brahmi-based script used mainly for writing Marathi. Modi was also used to write other regional languages such as Hindi, Gujarati, Kannada, Konkani, Persian, Tamil, and Telugu. According to an old legend, the Modi script was brought to India from Sri Lanka by Hemadri Pandit, known also as Hemadpant, who was the chief minister of Ramacandra, the last king of the Yadava dynasty, who reigned from 1271 to about 1309. Another tradition credits the creation of the script to Balaji Avaji, secretary of state to the late 17th-century Maratha king Shivaji Raje Bhonsle, also known as Chhatrapati Shivaji Maharaj. While the veracity of such accounts is difficult to ascertain, it is clear that Modi derives from the Nagari family of scripts and is a modification of the Nagari model intended for continuous writing.

Modi emerged as an administrative writing system in the 16th century before the rise of the Maratha dynasties. It was adopted by the Marathas as an official script beginning in the 17th century and was used in such a capacity in Maharashtra until the middle of the 20th century. In the 1950s the use of Modi was formally discontinued and the Devanagari script, known as "Balbodh," was promoted as the standard writing system for Marathi.

There are thousands of Modi documents preserved in South Asia and Europe. The majority of these are in various archives in Maharashtra, while smaller collections are kept in Denmark and other countries, because of European presence in Tanjore, Pondicherry, and other regions in South Asia through the 19th century. The earliest extant Modi document dates from the early 17th century. While the majority of Modi documents are official letters, land records, and other administrative documents, the script was also used in education, journalism, and other routine activities before the 1950s. Printing in Modi began in the early 19th century after Charles Wilkins cut the first metal fonts for the script in Calcutta. Newspapers were published in Modi; primers were produced to teach the script in schools, and various personal papers and diaries were kept in the script.

Structure. Modi is a Brahmi-based script related to Devanagari. It is written from left to right. In general, the rules for Devanagari rendering also apply to Modi (see *Section 12.1, Devanagari*). However, one characteristic feature of Modi is a large number of context-dependent forms of consonants and vowel-signs. Shaping and glyph substitutions for these contextual forms are managed in the font.

Vowel Letters. Generally, the distinction between regular and long forms of *i* and *u* is not preserved in Modi. The letter U+11603 MODI LETTER II may represent both *i* and *ī*, and U+11604 MODI LETTER U may be used for writing both *u* and *ū*. The same

can be said of the corresponding dependent vowel signs. Both regular and long forms appear in the Modi block, because they are attested in documentation about Modi.

The vocalic letters in the range U+11635..U+11638 are included in the encoding, but are not in modern use, as is the case in other Indic scripts. Modi *vocalic r* may alternatively be written as the sequence <U+11628 MODI LETTER RA, U+11632 MODI VOWEL SIGN II> *rī*.

Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 15-8* shows the letters that can be analyzed, the single code point that should be used to represent them in text, and the sequence of code points resulting from analysis that should not be used.

Table 15-8. Modi Vowel Letters

For	Use	Do Not Use
ॳ	1160A	<11600, 11639>
ॴ	1160B	<11600, 1163A>
ॵ	1160C	<11601, 11639>
ॶ	1160D	<11601, 1163A>

Rendering. Many of the consonant-vowel and consonant-consonant combinations in Modi involve special contextual forms of the consonant or vowel-sign or both. These are rendered by means of contextual rules in the font, using specially shaped and positioned glyph pieces or preformed ligatures.

Consonant Clusters Involving ra. A number of contextual forms are used for U+11628 ‌ MODI LETTER RA. Some of these are similar to the use of *ra* in Devanagari. As the first consonant in a cluster it is generally rendered as a *repha*; however, Modi also uses the *eyelash ra* in place of *repha* in certain native Marathi contexts. As in Devanagari, the *eyelash ra* is produced using the sequence <U+11628 ‌ MODI LETTER RA, U+1163F ◌ MODI SIGN VIRAMA, U+200D ⟨ZW⟩ ZERO WIDTH JOINER>.

Non-initial *ra* in conjuncts is typically rendered using one of two subjoined forms; however, some conjuncts with *ra* are represented as distinct ligatures. The most common of these is the conjunct ‌ represented by the sequence <U+1161D ‌ MODI LETTER TA, U+1163F ◌ MODI SIGN VIRAMA, U+11628 ‌ MODI LETTER RA>. Sequences of *ra* following some other consonants, such as <*ka, ra*>, <*ka, -aa, ra*>, or <*sa, ra*> are also displayed by distinct ligatures, as shown in *Figure 15-2*. The sequence of initial *ra* followed by the rounded consonants *kha*, *dha*, or *ha*, may also appear with distinct ligatures.

Figure 15-2. Modi Shaping for *ra*

Unusually, the shape of *ra* is also influenced at the word level, depending upon the characters in the preceding syllable. See the last example in *Figure 15-2*. This influence on the shape of *ra* may even occur preceding punctuation; in certain environments, *ra* following a *danda* or *double danda* is written using a special contextual form. For example:

U+11642 ‖ *double danda* + U+11628 ꣣ *ra* → ‖

To produce this behavior, the *danda* and *double danda* characters in the Modi block should be used instead of the ones in the Devanagari block.

Punctuation and Word Boundaries. Traditionally, word boundaries are not marked in Modi because it is an administrative script, characterized by the practice of rapid writing without lifting the pen. Paragraph and other section boundaries are, however, indicated in some Modi documents through the use of whitespace. Modern practice uses spaces and various punctuation conventions, including *danda* and Western punctuation marks. Some printed books use a period instead of a *danda* to indicate a sentence boundary.

Various Signs. Nasalization is indicated by U+1163D MODI SIGN ANUSVARA, and abbreviations are indicated using U+11643 MODI ABBREVIATION SIGN. U+1163E MODI SIGN VISARGA represents an allophone of *ra* or *sa* at word-final position in Sanskrit orthography. U+11640 MODI SIGN ARDHACANDRA is used for transcribing sounds used in English names and loanwords.

U+11644 MODI SIGN HUVA is written as an invocation in several Modi documents. It is derived from the Arabic *huwa*.

Currency values are written using U+A838 NORTH INDIC RUPEE MARK.

Numbers. Modi has a full set of decimal digits. Several number forms and unit marks are used for writing Modi and are represented using characters in the Common Indic Number Forms block. They include the base-16 fraction signs U+A830..U+A835. The absence of intermediate units is indicated by U+A837 NORTH

INDIC PLACEHOLDER MARK, which is called *ali* in Marathi. U+A836 NORTH INDIC QUARTER MARK is used for representing *anna* values.

15.13 Nandinagari

15.13.1 Nandinagari: U+119A0–U+119FF

Nandinagari is a Brahmi-based script that was used in southern India between the 11th and 19th centuries for manuscripts and inscriptions in Sanskrit in south Maharashtra, Karnataka and Andhra Pradesh. It is related to Devanagari, and was the official script of the Vijayanagara kingdom of southern India (1336–1646 CE). There are numerous manuscripts and inscriptions containing Nandinagari text. This script was also used for writing Kannada in Karnataka.

Structure. With minor historical exceptions, Nandinagari is an *abugida* written from left to right where there is a consonant plus an inherent vowel (usually the sound /a/), similar to Devanagari. The absence of the inherent vowel is frequently marked with a virama. The virama sign that suppresses the inherent vowel of the consonant is a combining character.

Headstrokes. These are an inherent feature of Nandinagari letters, but their behavior differs from headstrokes in modern Devanagari. Headstroke connections in Nandinagari generally are restricted to an *aksara* (orthographic syllable) and do not extend to neighboring syllables. The headstroke connects vowel or consonant letters and spacing dependent vowels of an *aksara*, while spaces separate individual *aksaras*.

Vowels. There are 12 vowel letters in the range U+119A0..U+119AD and 11 dependent vowel signs in the range U+119D1..U+119DD. U+119D2 NANDINAGARI VOWEL SIGN I is positioned at the top-left edge of letters that have headstrokes. For other letters U+119D2 hangs above the top-left portion of the body. However, the style of writing the sign varies considerably, particularly in handwriting.

Consonants. There are 35 consonant letters. U+119D0 NANDINAGARI LETTER RRA appears to have been introduced in the 11th century for transcribing the Kannada letter RRA, and is not part of the traditional repertoire of Nandinagari.

Virama. U+119E0 NANDINAGARI SIGN VIRAMA has two functions, similar to the corresponding Devanagari character. Used as a halanta, it marks the absence of the inherent vowel of a consonant letter. U+119E0 is also a format character used to produce conjuncts.

Vowel Modifiers. U+119DE NANDINAGARI SIGN ANUSVARA indicates nasalization. It is placed to the right of a base letter or right-side vowel sign. U+119DF NANDINAGARI SIGN VISARGA represents post-vocalic aspiration in words of Sanskrit origin.

Other Signs. U+119E1 NANDINAGARI SIGN AVAGRAHA marks the elision of word-initial *a* in Sanskrit as a result of *sandhi*. The auspicious sign U+119E2 NANDINAGARI SIGN SIDDHAM indicates an invocation at the beginning of documents.

Punctuation. U+119E3 NANDINAGARI HEADSTROKE is used as a sign of spacing or joining a word. It may connect a word that is broken on account of imperfections on a writing surface. U+119E3 can also serve as a gap filler. Nandinagari uses the danda and double danda marks encoded in the Devanagari block.

Digits. The Nandinagari digits are glyph variants of the Kannada digits U+0CE6..U+0CEF. No script specific digits are encoded for Nandinagari.

15.14 Grantha

15.14.1 Grantha: U+11300–U+1137F

The Grantha script descends from Brahmi. The modern form is chiefly used to write the Sanskrit language, including Vedic Sanskrit. It is used primarily in Tamil Nadu, and to a lesser extent in Sri Lanka and other parts of South India.

The Grantha script is frequently mixed with the Tamil script to write Sanskrit words. Grantha has also been used to write the Sanskrit words of Tamil Manipravalam—a mixed Sanskrit-Tamil language—though this usage has become rare. In addition, Grantha characters may occasionally be employed with the Tamil script in the writing systems of minority languages of southern India.

Historically, intermediate forms which gave rise to the Grantha script are attested as of the fourth century CE. The earliest examples are found in inscriptions of the early Pallava kings who ruled over parts of what is currently northern Tamil Nadu and southern Andhra Pradesh. Modern Grantha, which this encoding represents, belongs to the period after the thirteenth century CE.

Modern Grantha is frequently used by Tamil speakers to represent Sanskrit because Grantha's large set of letters can represent all the sounds of Sanskrit without the use of diacritical marks. The Tamil script has a smaller repertoire of letters that requires diacritical marks to represent Sanskrit directly. This use of diacritical marks often leads to confusion regarding the pronunciation of Sanskrit when written in the Tamil script.

15.14.2 Rendering Grantha

Although the Grantha script is visually similar to Tamil, its structure is similar to other Indic scripts that are used to write Sanskrit. Written Sanskrit requires support for stacked consonant structures.

Consonant Clusters. Some consonant clusters are stacks, some consonant structures are a combination of ligatures and stacks, and some are just ligatures. Ligatures are often used instead of stacks, and consonant clusters are frequently written as a combination of ligatures and stacking.

The typical stack height found in print in non-Vedic Sanskrit is two elements, but it is three in Vedic Sanskrit. Stacks, like ligatures, are equivalent to single consonants for the purpose of application of vowel signs.

Instances requiring more than three elements in a stack require special handling. In these cases, the initial elements are pushed out of the consonant stack and may form

their own stacks. Such special cases are illustrated in *Figure 15-3*. In this situation, a single phonological consonant cluster followed by a vowel may be represented by more than one orthographic cluster.

Figure 15-3. Splitting Large Conjunct Stacks in Grantha

two elements	→	two-level stack
three elements	→	three-level stack
four elements	→	vowelless element + three-level stack
five elements	→	vowelless two-level stack + three-level stack
six elements	→	vowelless three-level stack + three-level stack

Virama. Grantha follows the same *virama* model as Telugu and Kannada, in which the sequence *consonant + virama* should be rendered as the vowelless form of the consonant in the desired orthographic style. For example, in the prevalent ortho-graphic style used in modern printing, *ta*, *na*, and *ma* consistently fuse with the virama; *ra* and *la* superficially connect with it, and the virama stands apart for all other consonants, as shown in *Table 15-9*.

Table 15-9. Rendering of Explicit Virama Forms in Grantha

Fused					
ta + virama	ௐ	+	○	→	ௐ
na + virama	ௐ	+	○	→	ௐ
ma + virama	ௐ	+	○	→	ௐ
Connected					
ra + virama	ௐ	+	○	→	ௐ
la + virama	ௐ	+	○	→	ௐ
Unconnected					
ka + virama	ௐ	+	○	→	ௐ
tta + virama	ௐ	+	○	→	ௐ

These visual distinctions in the rendering of explicit viramas also apply to the various ligated conjuncts of Grantha.

Vowels. There are two forms of the *au* vowel sign: U+11357 GRANTHA AU LENGTH MARK is the modern one-part form, while the two-part form U+1134C GRANTHA VOWEL SIGN AU, is somewhat archaic, but is found in manuscripts.

Only two vowel signs touch their base consonant in printed Grantha: U+1133F GRANTHA VOWEL SIGN I and U+11340 GRANTHA VOWEL SIGN II. U+11347 GRANTHA VOWEL SIGN EE and U+11348 GRANTHA VOWEL SIGN AI are rendered to the left of their base. U+1134B GRANTHA VOWEL SIGN OO and the archaic U+1134C GRANTHA

VOWEL SIGN AU are two-part vowels with one part placed to the left of the base and one part to the right. All other vowel signs are placed to the right of the base.

Manuscripts written in Grantha will show archaic ligatures of consonants with vowel signs. The vowel signs U+11362 GRANTHA VOWEL SIGN VOCALIC L and U+11363 GRANTHA VOWEL SIGN VOCALIC LL are sometimes placed below and sometimes placed to the right of the base consonant. In contemporary printing practice, vowel signs are placed to the right.

Signs. Grantha uses the *pluta* sign to denote vowel lengthening. The *pluta* is not in current use, but it is found in Vedic manuscripts. The *nukta* is not used to write Sanskrit, but is used to transcribe words from other languages, such as Irula.

Cantillation Marks. Grantha uses a number of cantillation marks to represent tone, stress, and breathing in Vedic texts. These marks include the twelve marks encoded in the Grantha block in the range from U+11366..U+11374, and many encoded in other blocks as well, including those listed in *Table 15-10*.

Table 15-10. Additional *Svara* Marks used in Grantha

Generic Vedic Accents
0951 DEVANAGARI STRESS SIGN UDATTA
0952 DEVANAGARI STRESS SIGN ANUDATTA
Samavedic Marks
1CD0 VEDIC TONE KARSHANA
1CD2 VEDIC TONE PRENKHA
1CD3 VEDIC SIGN NIHSHVASA
20F0 COMBINING ASTERISK ABOVE
Additional Marks
1CF2 VEDIC SIGN ARDHAVISARGA
1CF3 VEDIC SIGN ROTATED ARDHAVISARGA
1CF4 VEDIC TONE CANDRA ABOVE
1CF8 VEDIC TONE RING ABOVE
1CF9 VEDIC TONE DOUBLE RING ABOVE

These nonspacing marks are normally applied to independent vowels, to consonants with an inherent vowel, and to consonants with vowel signs. Sometimes they are also applied to dead consonants which are displayed with a visible *virama*.

The preferred placement of *svara* marks in Grantha is horizontally centered relative to the syllable. These marks should not extend beyond the horizontal span of the base syllable. The *svara* marks can be applied to either syllables or digits, and used in combination with each other.

Punctuation. *Danda* and *double danda* marks used with Grantha are found in the Devanagari block; see *Section 12.1, Devanagari*.

Line Breaking. Line breaks may occur after every orthographic syllable. Hyphens are not used.

Numbers. Grantha makes use of the Tamil digits U+0BE6 through U+0BEF, as well as the Tamil historical numerals for ten, one hundred, and one thousand at U+0BF0..U+0BF2. Grantha also uses some numbers and symbols from the Tamil Supplement block in the range U+11FC0..U+11FFF, that contains a set of historic fractions and other symbols.

15.15 Dives Akuru

15.15.1 Dives Akuru: U+11900–U+1195F

Dives Akuru or Divehi Akuru was a script used to write the Dhivehi language on the Maldives from the 9th to the 20th centuries. Dives Akuru literally means "islanders' letters." The script is most closely related to a medieval form of the Sinhala script. In the 18th century, the Thaana script appeared alongside Dives Akuru. By the turn of the 19th century, Thaana had replaced Dives Akuru as the regular script for Dhivehi. However, individuals and scholars continued to study and use Dives Akuru into the 20th century.

Today, the script style from the 12th to the 14th centuries is termed *evēla akuru*, while the script style after the 14th century is called *dives akuru*. Both styles are unified in the Dives Akuru repertoire. Because no traditional documentation of the letter inventory exists for the script, the repertoire is based on texts found on copper plates, paper, and wooden boards, with the broadest repertoire found in the *evēla akuru* documents. The different styles and specific variants of characters should be handled through fonts.

Structure. Dives Akuru is an abugida, written left to right. Like other Brahmi-derived scripts, each consonant letter contains an inherent vowel *a*. To indicate the bare consonant, U+1193D DIVES AKURU SIGN HALANTA is used. Consonant clusters are typically rendered by conjuncts.

Vowels. Independent vowels are represented either by the distinctive vowel letters (U+11900..U+11909) or by an orthographic syllable composed of U+11925 DIVES AKURU LETTER YA, which acts as a vowel carrier, and the dependent vowel sign.

Conjuncts. In general, consonant clusters are rendered as conjuncts in Dives Akuru. Most conjuncts consist of clusters of two consonants, but conjuncts with up to three consonants are attested. Four conjoining forms are encoded atomically: two are cluster-initial (U+11941 DIVES AKURU INITIAL RA and U+1193F DIVES AKURU PREFIXED NASAL SIGN) and two are syllable medial (U+11940 DIVES AKURU MEDIAL YA and U+11942 DIVES AKURU MEDIAL RA).

The conjunct structure visually consists of letters that are joined in a distinctive ligature or as a touching ligature. A touching ligature is produced when writing letters together without spaces, so they touch at adjacent edges.

The script uses U+1193E DIVES AKURU VIRAMA to create conjuncts, but no virama is required when using the four atomically-encoded conjoining form characters. Vowel letters may participate in clustering, especially when the second member of the

cluster appears in a touching ligature, right after the word boundary. Vowel signs are encoded after the conjunct.

Halanta. In Dives Akuru, U+1193D DIVES AKURU SIGN HALANTA has multiple functions. As a vowel-killer, the *halanta* generally attaches to the right-hand side of a letter, and forms a ligature with its base. While the *halanta* typically suppresses a consonant's inherent vowel, in some cases the sequence <consonant, halanta> is pronounced as a syllable with /u/. In addition, the consonants *ka*, *na*, *tta*, and *ta* with an attached *halanta* may, in certain cases, be rendered as superscripts.

Nasalization Signs. Post-vocalic nasalization is indicated using U+1193C DIVES AKURU SIGN CANDRABINDU and U+1193B DIVES AKURU SIGN ANUSVARA.

Nukta. U+11943 DIVES AKURU SIGN NUKTA is used to transcribe sounds that are not native to Dhivehi. The *nukta* is written below the letter which most closely approximates the foreign sound.

Digits. Script-specific digits are used for Dives Akuru. They are encoded in the range U+11950..U+11959.

Punctuation. Three marks of punctuation are encoded for representing Dives Akuru text. A script-specific double danda is encoded at U+11944. Another punctuation mark, U+11945 DIVES AKURU GAP FILLER, is used to fill space at the ends of lines or to signify the end of a document. U+11946 DIVES AKURU END OF TEXT MARK appears at the end of a document, and is often accompanied by U+11945 DIVES AKURU GAP FILLER.

Line Breaking. Line breaks may occur after any orthographic syllable. Hyphens are not used. Fillers may be used to fill space at the ends of lines, as described in the description of punctuation.

15.16 Ahom

15.16.1 Ahom: U+11700–U+1174F

The Ahom script is used in northeast India, primarily to write the Tai Ahom language. The oldest surviving Ahom text is the "Snake Pillar" inscription which was inscribed in the time of King Siuw Hum Miung (1497-1539). The script also appears on other stone inscriptions, coins, brass plates and a large corpus of manuscripts. Although the use of the Tai Ahom language declined in the late 17th century, traditional priests used the language and the Ahom script in their religious practices throughout the 19th century.

Modern use of the Ahom script is considered to have begun in 1920 with the publication of an Ahom-Assamese-English dictionary. This was followed by publication of other dictionaries, word lists, and primers. The publication of Ahom texts has progressed more rapidly in recent decades, thanks to the availability of computers. Today there are large numbers of books published in Assam that contain some Ahom content.

Structure. Like most other Brahmi-derived scripts, Ahom is an *abugida*, for which consonant letters are associated with an inherent vowel "a". The encoding also includes three medial consonants, in the range U+1171D..U+1171F, which follow and graphically attach to an initial consonant letter. In addition, Ahom has a visible virama that functions as a vowel killer, U+1172B AHOM SIGN KILLER. The use of the killer is only obligatory in modern Ahom.

Vowels. Ahom has no independent vowels, but instead uses U+11712 AHOM LETTER A followed by the corresponding dependent vowel sign (or signs).

Syllabic Structure. Ahom has closed syllables, and optional medials may occur after initial consonants. Vowels can occur in sequences of U+11712 AHOM LETTER A and dependent vowel signs, or a series of dependent vowel signs. Final consonants take U+1172B AHOM SIGN KILLER.

Numerals. The original Ahom numeral system was not a decimal radix system; however, in modern use a digit zero has been added, and the digits can be used to express decimal radix numerals. In traditional use, the digits may also be mixed with word spellings when writing out numbers.

The forms of the Ahom digits are derived from several sources. U+11732 AHOM DIGIT TWO is visually identical to U+11701 AHOM LETTER KHA and probably derives from it. The digits 3, 4, and 5 are usually expressed by the Ahom words for those numbers spelled out. U+1173B AHOM NUMBER TWENTY is also just the Ahom word for 20 spelled out.

Punctuation. Ahom uses two punctuation characters which function similarly to *dandas*: U+1173C ᴀʜᴏᴍ sɪɢɴ sᴍᴀʟʟ sᴇᴄᴛɪᴏɴ and U+1173D ᴀʜᴏᴍ sɪɢɴ sᴇᴄᴛɪᴏɴ. The script also uses a paragraph mark, U+1173E ᴀʜᴏᴍ sɪɢɴ ʀᴜʟᴀɪ, and a symbol that indicates an exclamation, U+1173F ᴀʜᴏᴍ sʏᴍʙᴏʟ ᴠɪ.

Modern Ahom uses spaces to indicate word boundaries. This convention is seen in some early Ahom manuscripts, but is not consistent in the early material.

Variant Forms. A number of variant letterforms are found in manuscripts, but are no longer used in modern Ahom. Specific characters are encoded to represent the historic variants of *ta*, *ga*, *ba*, and the medial ligating *ra*.

15.17 Sora Sompeng

15.17.1 Sora Sompeng: U+110D0–U+110FF

The Sora Sompeng script is used to write the Sora language. Sora is a member of the Munda family of languages, which, together with the Mon-Khmer languages, makes up Austro-Asiatic.

The Sora people live between the Oriya- and Telugu-speaking populations in what is now the Odisha-Andhra border area.

Sora Sompeng was devised in 1936 by Mangei Gomango, who was inspired by the vision he had of the 24 letters. The script was promulgated as part of a comprehensive cultural program, and was offered as an improvement over IPA-based scripts used by linguists and missionaries, and the Telugu and Oriya scripts used by Hindus. Sora Sompeng is used in religious contexts, and is published in a variety of printed materials.

Encoding Structure. The Sora Sompeng script is an abugida. The consonant letters contain an inherent vowel. There are no conjunct characters for consonant clusters, and there is no visible vowel killer to show the deletion of the inherent vowel. The reader must determine the presence or absence of the inherent schwa based on recognition of each word. The character repertoire does not match the phonemic repertoire of Sora very well.

U+110E4 SORA SOMPENG LETTER IH is used for both [i] and [ɨ], and U+110E6 SORA SOMPENG LETTER OH is used for both [o] and [ɔ], for instance. The glottal stop is written with U+110DE SORA SOMPENG LETTER HAH, and the sequence of U+110DD SORA SOMPENG LETTER RAH and U+110D4 SORA SOMPENG LETTER DAH is used to write retroflex [ɽ]. There is also an additional "auxiliary" U+110E8 SORA SOMPENG LETTER MAE used to transcribe foreign sounds.

Character Names. Consonant letter names for Sora Sompeng are derived by adding [aʔa] (written *ah*) to the consonant.

Punctuation. Sora Sompeng uses Western-style punctuation.

Line Breaking. Letters and digits behave as in Latin and other alphabetic scripts.

15.18 Tulu-Tigalari

15.18.1 Tulu-Tigalari: U+11380–U+113FF

Tulu-Tigalari was used to primarily write Sanskrit religious texts, but a small number of Tulu and Kannada language texts are written using this script. Tulu-Tigalari is influenced by scripts such as medieval Grantha, Vatteluttu, and Telugu-Kannada. The script has been used since at least 1250 CE.

Structure. The structure of the Tulu-Tigalari script is similar to that of other Brahmic scripts. Each consonant letter contains an inherent vowel *a*. It is an *abugida* that makes use of a virama. The script is written from left to right.

Consonant Letters. There are 36 consonants in Tulu-Tigalari, encoded in the range U+11392..U+113B5. Two of the consonants represent Dravidian sounds and are quite rare: U+113B4 ಀ TULU-TIGALARI LETTER RRA and U+113B5 ಀ TULU-TIGALARI LETTER LLLA.

Independent Vowels. Tulu-Tigalari has 14 independent vowels, encoded in the range U+11380..U+11391. These include the two diphthongs, U+1138E ಀಀ TULU-TIGALARI LETTER AI and U+11391 ಀಀ TULU-TIGALARI LETTER AU. Similarly to many other Indic scripts, these 14 vowels are encoded atomically.

The alternate or rare forms of vowel letters *i*, *u*, *vocalic r*, *vocalic rr* and *vocalic l* should be handled as sequences, as shown in <u>*Figure 15-4*</u>.

Figure 15-4. Rare Forms of Tulu-Tigalari Vowels

11382 ಀ + 113B8 ಀ → ಀ

11382 ಀ + 113BC ಀ → ಀ

11384 ಀ + 113BC ಀ → ಀ

11384 ಀ + 113C9 ಀ → ಀ

113D1 ಀ + 11386 ಀ → ಀ

113D1 ಀ + 11387 ಀ → ಀ

113D1 ಀ + 11388 ಀ → ಀ

Dependent Vowel Signs. All independent vowels except for U+11380 ಀಀ TULU-TIGALARI LETTER A have a corresponding dependent vowel sign, encoded in the

range U+113B8..U+113C8. These signs are positioned to the left, right, or below consonants and conjuncts, replacing the inherent vowel *a*.

Four Tulu-Tigalari vowel signs are rendered as ligatures which appear below the consonant or conjunct and ligate to the right. These are U+113BB ఒ TULU-TIGALARI VOWEL SIGN U, U+113BC ఒ TULU-TIGALARI VOWEL SIGN UU, U+113BD ఒ TULU-TIGALARI VOWEL SIGN VOCALIC R, and U+113BE ఒ TULU-TIGALARI VOWEL SIGN VOCALIC RR.

Additionally, the vowel signs *u* and *uu* change their shape depending on the consonant or conjunct they combine with. Some consonant plus vowel sign sequences can have alternate forms. A few of the many possible ligatures are shown in *Figure 15-5*.

Figure 15-5. Examples of Ligatures in Tulu-Tigalari

౧ *pa* + ఒ *sign u* → ౧౸ *pu*

౬౦ *cha* + ఒ *sign u* → ౬౦ఒ *chu*

౬ *ka* + ఒ *sign u* → ౬ఒ *ku*

The Tulu-Tigalari script encodes several two-part vowel characters. U+113C7 ౧౦౧ TULU-TIGALARI VOWEL SIGN OO and U+113C8 ౧౦౧ TULU-TIGALARI VOWEL SIGN AU are split vowel signs that appear both before and after a character or conjunct. For a detailed discussion of the use of two-part vowels, see "Two-Part Vowels" in *Section 12.6, Tamil*.

Canonical Equivalences. Some of the independent and dependent vowels can be visually analyzed as consisting of multiple parts corresponding to the shapes of other vowels, as shown in *Figure 15-6*. These multipart vowels have canonical decompositions, so that the atomic characters and the corresponding sequences are canonical equivalents. The atomic characters are the typical representation used when generating text.

Figure 15-6. Tulu-Tigalari Canonical Sequences

11383 ౬౦   ≡   11382 ౬   +   113C9 ౦౧

11385 ౬ఒ   ≡   11384 ౬   +   113BB ఒ

1138E ౧౦౬౦   ≡   1138B ౧౦౬   +   113C2 ౧౦

11391 ౬౦౧   ≡   11390 ౬౦   +   113C9 ౦౧

113C5 ౧౦౧౦   ≡   113C2 ౧౦   +   113C2 ౧౦

113C7 ౧౦౧   ≡   113C2 ౧౦   +   113B8 ౧౧

113C8 ౧౦౧   ≡   113C2 ౧౦   +   113C9 ౦౧

Various Signs. U+113C9 ◌ TULU-TIGALARI AU LENGTH MARK is not used on its own as a complete vowel sign. This mark is used to render the two-part *vowel sign au* and the *letter ii*.

The U+113CA ◌ TULU-TIGALARI SIGN CANDRA ANUNASIKA mark is analogous to the *candrabindu* found in other Indic scripts. It can combine with all letters and vowel signs.

A pure nasal sound is represented by U+113CC ◌ TULU-TIGALARI SIGN ANUSVARA. U+113CD ◌ TULU-TIGALARI SIGN VISARGA indicates a voiceless glottal fricative. Both *anusvara* and *visarga* are rendered to the right of the affected character.

A spacing mark, U+113B7 ꠯ TULU-TIGALARI SIGN AVAGRAHA, is used when rendering Sanskrit texts. U+113D3 ꠯ TULU-TIGALARI SIGN PLUTA is used to denote vowel lengthening.

U+113E1 ◌ TULU-TIGALARI VEDIC TONE SVARITA and U+113E2 ◌ TULU-TIGALARI VEDIC TONE ANUDATTA are tone marks used in the representation of Vedic text in Tulu-Tigalari. These two combining marks are centered directly above or below a cluster, respectively.

Viramas and Conjoiner. U+113CE ◌ TULU-TIGALARI SIGN VIRAMA is an inherent vowel killer, and is also used in combination with other vowels to represent the Tulu vowels *ŭ* [u] and *ū̆* [u:]. Consequently, it can appear after vowel signs. *Figure 15-7* shows the usual convention. The virama always appears at the end of and to the top right of a cluster.

Figure 15-7. Examples of Vowels *ŭ* and *ū̆* in Tulu-Tigalari

꠯ *a*	+ ◌ *virama*	→ ꠯	*ŭ*
꠯ *ka*	+ ◌ *virama*	→ ꠯	*k(ŭ)*
꠯ *aa*	+ ◌ *virama*	→ ꠯	*ū̆*
꠯ *ka*	+ ◌ *sign aa* + ◌ *virama*	→ ꠯	*kū̆*

Unlike in Devanagari or Kannada, viramas in Tulu-Tigalari do not form conjuncts. Instead, U+113D0 ◌ TULU-TIGALARI CONJOINER is used for the formation of conjuncts. Consonants can combine horizontally, vertically, or have a combination of both, as shown in *Figure 15-8*. There is a preference for horizontal ligatures (where attested) over stacked vertical conjuncts. U+113CF ꠯ TULU-TIGALARI SIGN LOOPED VIRAMA is used to form the looped virama ligatures. It is only attested for *ka*, *ga*, *tta*, *ta*, and *na* (and some conjuncts that end with these consonants). The looped virama is tightly bound to the preceding character and does not apply at a syllable level. Conjunct sequences that end with a looped virama are rare.

Figure 15-8. Conjuncts and Viramas in Tulu-Tigalari

ಕ *ka* + ್ *virama* + ಕ *ka* → ಕ್ಕ *k(ŭ)ka*

ಕ *ka* + ⊙ *conjoiner* + ಕ *ka* → ಕ್ಕ *kka*

ಕ *ka* + ⊙ *conjoiner* + ಖ *kha* → ಕ್ಖ *kkha*

ಕ *ka* + ್ *looped virama* + ಕ *ka* → ಕ್ಕ *kka*

ಕ *ka* + ್ *looped virama* + ⊙ *conjoiner* + ಕ *ka* → ಕ್ಕ *kka*

ಕ *ka* + ⊙ *conjoiner* + ಕ *ka* + ್ *looped virama* → ಕ್ಕ *kk*

The common way of representing gemination is by conjuncts. However, a gemination mark is also used in many manuscripts. The U+113D2 ಂ TULU-TIGALARI GEMINATION MARK is placed after the base letter. Other combining vowel signs are added after the gemination mark.

Repha. U+113D1 ⊡ TULU-TIGALARI REPHA is used to indicate a *ra* without the inherent vowel that precedes a vowel, consonant, or semi-vowel. The *repha* is shown in the code charts with a dashed box to emphasize its unusual behavior in interacting with the following consonant.

The *repha* most commonly displays as a short vertical line above the base consonant or conjunct, as shown in .

Figure 15-9. *Repha* Rendered as a Short Vertical Line

⊡ *repha* + ಕ *ka* → ಕ̇ *rka*

However, *repha* typically ligates with *ma*, *ya*, or *va*, as shown in .

Figure 15-10. *Repha* Ligating with *ma*, *ya*, or *va*

⊡ *repha* + ಮ *ma* → ಮ̇ *rma*

⊡ *repha* + ಯ *ya* → ಯ̇ *rya*

⊡ *repha* + ವ *va* → ವ̇ *rva*

When *repha* and a virama co-occur in a syllable, the *repha* visually ligates with the virama, as shown in .

Figure 15-11. *Repha* Ligating with Virama

⬚ *repha* + ⟨ka⟩ *ka* + ⟨virama⟩ *virama* → ⟨rk⟩ *rk(ŭ)*

Digits. The Kannada digits U+0CE6..U+0CEF should be employed to represent digits in Tulu-Tigalari.

Punctuation. Tulu-Tigalari has script-specific forms of the *danda* and *double danda* punctuation marks: U+113D4 ǀ TULU-TIGALARI DANDA and U+113D5 ǁ TULU-TIGALARI DOUBLE DANDA.

U+113D7 ಃ TULU-TIGALARI SIGN OM PUSHPIKA can either represent the *om* sound or it can be used as an indicator for beginnings, pauses, endings, or space fillers. Although *om pushpika* and U+113D8 ⊚ TULU-TIGALARI SIGN SHRII PUSHPIKA may superficially resemble the corresponding phonetic syllables, they are used as space fillers and for other decorative purposes.

Chapter 16

Southeast Asia-I

Thailand, Laos, Myanmar, Cambodia, Vietnam

This chapter documents the scripts of mainland Southeast Asia, also known as the Indochinese Peninsula.

Most scripts of Southeast Asia described in this chapter are written from left to right; many use no interword spacing but use spaces or marks between phrases. They are mostly *abugidas*, but with various idiosyncrasies that distinguish them from the scripts of South Asia.

Thai and Lao are the official scripts of Thailand and Laos, respectively, and are closely related. These scripts are unusual for Brahmi-derived scripts in the Unicode Standard, because for various implementation reasons they depart from logical order in the representation of consonant-vowel sequences. Vowels that occur to the left side of their consonant are represented in visual order before the consonant in a string, even though they are pronounced afterward.

Myanmar is the official script of Myanmar, and is used to write the Burmese language, as well as many minority languages of Myanmar and Northern Thailand. It has a mixed encoding model, making use of both a virama and a killer character, and having explicitly encoded medial consonants.

The Khmer script is used for the Khmer and related languages in the Kingdom of Cambodia.

The term "Tai" refers to a family of languages spoken in Southeast Asia, including Thai, Lao, and Shan. This term is also part of the name of a number of scripts encoded in the Unicode Standard. The Tai Le script is used to write the language of the same name, which is spoken in south central Yunnan (China). The New Tai Lue script, also known as Xishuangbanna Dai, is unrelated to the Tai Le script, but is also used in south Yunnan. New Tai Lue is a simplified form of the more traditional Tai Tham script, which is also known as Lanna. The Tai Tham script is used for the Northern Thai, Tai Lue, and Khün languages. The Tai Viet script is used for the Tai Dam, Tai Dón, and Thai Song languages of northwestern Vietnam, northern Laos,

and central Thailand. Unlike the other Tai scripts, the Tai Viet script makes use of a visual order model, similar to that for the Thai and Lao scripts.

Kayah Li is a relatively recently invented script, used to write the Kayah Li languages of Myanmar and Thailand. Although influenced by the Myanmar script, Kayah Li is basically an alphabet in structure.

Cham is a Brahmi-derived script used by the Austronesian language Cham, spoken in the southern part of Vietnam and in Cambodia. It does not use a virama. Instead, the encoding makes use of medial consonant signs and explicitly encoded final consonants.

Pahawh Hmong is an alphabetic script devised for writing the Hmong language in the latter half of the 20th century. Its development includes several revisions. The script is used by Hmong communities in several countries, including the United States and Australia.

Nyiakeng Puachue Hmong is a writing system created in the 1980s to write the White Hmong and Green Hmong languages. It is also called the Ntawv Txawjvaag or Cher-vang script, and was devised for use in the United Christians Liberty Evangelical church in the United States. The script is written from left to right, and is reported to be used in Laos, Thailand, Vietnam, France and Australia.

The Pau Cin Hau alphabet is a liturgical script of the Laipian religious tradition, which emerged in the Chin Hills region of present-day Chin State, Myanmar at the turn of the 20th century.

Hanifi Rohingya is an alphabetic script used to write the Rohingya language, an Indo-Aryan language spoken by one million people primarily in Myanmar and Bangladesh. The script was developed in the 1980s and shows Arabic influence in its general appearance and structure.

16.1 Thai

16.1.1 Thai: U+0E00–U+0E7F

The Thai script is used to write Thai and other Southeast Asian languages, such as Kuy, Lanna Tai, and Pali. It is a member of the Indic family of scripts descended from Brahmi. Thai modifies the original Brahmi letter shapes and extends the number of letters to accommodate features of the Thai language, including tone marks derived from superscript digits. At the same time, the Thai script lacks the conjunct consonant mechanism and independent vowel letters found in most other Brahmi-derived scripts. As in all scripts of this family, the predominant writing direction is from left to right.

Standards. Thai layout in the Unicode Standard is based on the Thai Industrial Standard 620-2529, and its updated version 620-2533.

Encoding Principles. In common with most Brahmi-derived scripts, each Thai consonant letter represents a syllable possessing an inherent vowel sound. For Thai, that inherent vowel is /o/ in the medial position and /a/ in the final position.

The consonants are divided into classes that historically represented distinct sounds, but in modern Thai indicate tonal differences. The inherent vowel and tone of a syllable are then modified by addition of vowel signs and tone marks attached to the base consonant letter. Some of the vowel signs and all of the tone marks are rendered in the script as diacritics attached above or below the base consonant. These combining signs and marks are encoded after the modified consonant in the memory representation.

Most of the Thai vowel signs are rendered by full letter-sized inline glyphs placed either before (that is, to the left of), after (to the right of), or *around* (on both sides of) the glyph for the base consonant letter. In the Thai encoding, the letter-sized glyphs that are placed before (left of) the base consonant letter, in full or partial representation of a vowel sign, are, in fact, encoded as separate characters that are typed and stored *before* the base consonant character. This encoding for left-side Thai vowel sign glyphs (and similarly in Lao and in Tai Viet) differs from the conventions for all other Indic scripts, which uniformly encode all vowels after the base consonant. The difference is necessitated by the encoding practice commonly employed with Thai character data as represented by the Thai Industrial Standard.

The glyph positions for Thai syllables are summarized in *Table 16-1*.

Table 16-1. Glyph Positions in Thai Syllables

Syllable	Glyphs	Code Point Sequence

ka	กะ	0E01 0E30
ka:	กา	0E01 0E32
ki	กิ	0E01 0E34
ki:	กี	0E01 0E35
ku	กุ	0E01 0E38
ku:	กู	0E01 0E39
ku'	กึ	0E01 0E36
ku':	กื	0E01 0E37
ke	เกะ	0E40 0E01 0E30
ke:	เก	0E40 0E01
kae	แกะ	0E41 0E01 0E30
kae:	แก	0E41 0E01
ko	โกะ	0E42 0E01 0E30
ko:	โก	0E42 0E01
ko'	เกาะ	0E40 0E01 0E32 0E30
ko':	กอ	0E01 0E2D
koe	เกอะ	0E40 0E01 0E2D 0E30
koe:	เกอ	0E40 0E01 0E2D
kia	เกีย	0E40 0E01 0E35 0E22
ku'a	เกือ	0E40 0E01 0E37 0E2D
kua	กัว	0E01 0E31 0E27
kaw	เกา	0E40 0E01 0E32
koe:y	เกย	0E40 0E01 0E22
kay	ไก	0E44 0E01
kay	ใก	0E43 0E01
kam	กำ	0E01 0E33
kri	กฤ	0E01 0E24

Rendering of Thai Combining Marks. The canonical combining classes assigned to tone marks (ccc = 107) and to other combining characters displayed above (ccc = 0) do not fully account for their typographic interaction.

For the purpose of rendering, the Thai combining marks above (U+0E31, U+0E34..U+0E37, U+0E47..U+0E4E) should be displayed outward from the base character they modify, in the order in which they appear in the text. In particular, a sequence containing <U+0E48 THAI CHARACTER MAI EK, U+0E4D THAI CHARACTER NIKHAHIT> should be displayed with the *nikhahit* above the *mai ek*, and a sequence containing <U+0E4D THAI CHARACTER NIKHAHIT, U+0E48 THAI CHARACTER MAI EK> should be displayed with the *mai ek* above the *nikhahit*.

This does not preclude input processors from helping the user by pointing out or correcting typing mistakes, perhaps taking into account the language. For example, because the string <*mai ek, nikhahit*> is not useful for the Thai language and is likely a typing mistake, an input processor could reject it or correct it to <*nikhahit, mai ek*>.

When the character U+0E33 THAI CHARACTER SARA AM follows one or more tone marks (U+0E48..U+0E4B), the *nikhahit* that is part of the *sara am* should be displayed below those tone marks. In particular, a sequence containing <U+0E48 THAI CHARACTER MAI EK, U+0E33 THAI CHARACTER SARA AM> should be displayed with the *mai ek* above the *nikhahit*.

Thai Punctuation. Thai uses a variety of punctuation marks particular to this script. U+0E4F THAI CHARACTER FONGMAN is the Thai bullet, which is used to mark items in lists or appears at the beginning of a verse, sentence, paragraph, or other textual segment. U+0E46 THAI CHARACTER MAIYAMOK is used to mark repetition of preceding letters. U+0E2F THAI CHARACTER PAIYANNOI is used to indicate elision or abbreviation of letters; it is itself viewed as a kind of letter, however, and is used with considerable frequency because of its appearance in such words as the Thai name for Bangkok. *Paiyannoi* is also used in combination (U+0E2F U+0E25 U+0E2F) to create a construct called *paiyanyai*, which means "et cetera, and so forth." The Thai *paiyanyai* is comparable to its analogue in the Khmer script: U+17D8 KHMER SIGN BEYYAL.

U+0E5A THAI CHARACTER ANGKHANKHU is used to mark the end of a long segment of text. It can be combined with a following U+0E30 THAI CHARACTER SARA A to mark a larger segment of text; typically this usage can be seen at the end of a verse in poetry. U+0E5B THAI CHARACTER KHOMUT marks the end of a chapter or document, where it always follows the *angkhankhu* + *sara a* combination. The Thai *angkhankhu* and its combination with *sara a* to mark breaks in text have analogues in many other Brahmi-derived scripts. For example, they are closely related to U+17D4 KHMER SIGN KHAN and U+17D5 KHMER SIGN BARIYOOSAN, which are themselves ultimately related to the *danda* and *double danda* of Devanagari.

Spacing. Thai words are not separated by spaces. Instead, text is laid out with spaces introduced at text segments where Western typography would typically make use of commas or periods. However, Latin-based punctuation such as comma, period, and colon are also used in text, particularly in conjunction with Latin letters or in formatting numbers, addresses, and so forth. If explicit word break or line break opportunities are desired—for example, for the use of automatic line layout algorithms—the character U+200B ZERO WIDTH SPACE should be used to place invisible marks for such breaks. The ZERO WIDTH SPACE is ignored when justification or letter spacing is applied to the text. See *Table 23-2*.

Thai Transcription of Pali and Sanskrit. The Thai script is frequently used to write Pali and Sanskrit. When so used, consonant clusters are represented by the explicit use of U+0E3A THAI CHARACTER PHINTHU (*virama*) to mark the removal of the inherent vowel. There is no conjoining behavior, unlike in other Indic scripts. U+0E4D THAI CHARACTER NIKHAHIT is the Pali *nigghahita* and Sanskrit *anusvara*. U+0E30 THAI CHARACTER SARA A is the Sanskrit *visarga*. U+0E24 THAI CHARACTER RU and U+0E26 THAI CHARACTER LU are vocalic /r/ and /l/, with U+0E45 THAI CHARACTER LAKKHANGYAO used to indicate their lengthening.

Patani Malay. The Patani Malay orthography makes use of additional diacritics. A line below a consonant indicates that its sound differs from Thai. The line below is represented using U+0331 COMBINING MACRON BELOW. Nasalization is indicated by U+0303 COMBINING TILDE. Glottalization is marked with the character U+02BC MODIFIER LETTER APOSTROPHE. The character U+02D7 MODIFIER LETTER MINUS SIGN indicates an elision between two vowel sequences.

In Thai script, use of marks from the Combining Diacritical Marks block, such as U+0331 COMBINING MACRON BELOW and U+0303 COMBINING TILDE, imposes additional constraints for rendering systems. This is because the canonical ordering of these marks with respect to Thai vowels and tone marks may put them in an order that requires rearrangement during rendering.

In particular, when used as a consonant diacritic, U+0331 COMBINING MACRON BELOW can occur with vowel signs U+0E38 THAI CHARACTER SARA U or U+0E39 THAI CHARACTER SARA UU. These vowel signs have a fixed-position canonical combining class of 103. A character sequence would normally be entered in the order *consonant + macron below + vowel sign*. However, in normalized text, these combining marks would be re-ordered, resulting in a sequence *consonant + vowel sign + macron below*. Thai rendering implementations must ensure that the vowel signs SARA U and SARA UU are less-closely bound to the consonant letter than consonant diacritics. In other words, SARA U and SARA UU must appear below COMBINING MACRON BELOW in normalized text, and not vice versa.

Likewise, Thai tone marks U+0E48..U+0E4B have a fixed-position canonical combining class of 107. If a combining mark such as U+0303 COMBINING TILDE is used as a vowel sign, then it can potentially occur with the tone marks. Characters would likely be entered in the order *consonant + tilde + tone*, but in normalized text these would be reordered as *consonant + tone + tilde*. Thai rendering implementations must ensure that the tone marks display above the combining tilde.

16.2 Lao

16.2.1 Lao: U+0E80–U+0EFF

The Lao language and script are closely related to Thai. The Unicode Standard encodes the characters of the Lao script in roughly the same relative order as the Thai characters.

Encoding Principles. Lao contains fewer letters than Thai because by 1960 it was simplified to be fairly phonemic, whereas Thai maintains many etymological spellings that are homonyms. Unlike in Thai, Lao consonant letters are conceived of as simply representing the consonant sound, rather than a syllable with an inherent vowel. The vowel [a] is always represented explicitly with U+0EB0 LAO VOWEL SIGN A.

Punctuation. Regular word spacing is not used in Lao; spaces separate phrases or sentences instead.

Glyph Placement. The glyph placements for Lao syllables are summarized in [Table 16-2](#).

Table 16-2. Glyph Positions in Lao Syllables

Syllable	Glyphs	Code Point Sequence
ka	ກະ	0E81 0EB0
ka:	ກາ	0E81 0EB2
ki	ກິ	0E81 0EB4
ki:	ກີ	0E81 0EB5
ku	ກຸ	0E81 0EB8
ku:	ກູ	0E81 0EB9
ku'	ກຶ	0E81 0EB6
ku':	ກື	0E81 0EB7
ke	ເກະ	0EC0 0E81 0EB0
ke:	ເກ	0EC0 0E81
kae	ແກະ	0EC1 0E81 0EB0
kae:	ແກ	0EC1 0E81
ko	ໂກະ	0EC2 0E81 0EB0
ko:	ໂກ	0EC2 0E81
ko'	ເກາະ	0EC0 0E81 0EB2 0EB0
ko':	ກໍ	0E81 0ECD
koe	ເກິ	0EC0 0E81 0EB4
koe:	ເກີ	0EC0 0E81 0EB5

kia	ເກັຽງ ເກຍ	0EC0 0E81 0EB1 0EBD 0EC0 0E81 0EA2
ku'a	ເກືອ	0EC0 0E81 0EB7 0EAD
kua	ກົວ	0E81 0EBB 0EA7
kaw	ເກົາ	0EC0 0E81 0EBB 0EB2
koe:y	ເກີຽງ ເກີຍ	0EC0 0E81 0EB5 0EBD 0EC0 0E81 0EB5 0EA2
kay	ໄກ	0EC4 0E81
kay	ໃກ	0EC3 0E81
kam	ກຳ	0E81 0EB3

Additional Letters. A few additional letters in Lao have no match in Thai:

U+0EBB LAO VOWEL SIGN MAI KON

U+0EBC LAO SEMIVOWEL SIGN LO

U+0EBD LAO SEMIVOWEL SIGN NYO

The preceding two semivowel signs are the last remnants of the system of subscript medials, which in Myanmar retains additional distinctions. Myanmar and Khmer include a full set of subscript consonant forms used for conjuncts. Thai no longer uses any of these forms; Lao has just the two.

Rendering of Lao Combining Marks. The canonical combining classes assigned to tone marks (ccc = 122) and to other combining characters displayed above (ccc = 0) do not fully account for their typographic interaction.

For the purpose of rendering, the Lao combining marks above (U+0EB1, U+0EB4..U+0EB7, U+0EBB, U+0EC8..U+0ECD) should be displayed outward from the base character they modify, in the order in which they appear in the text. In particular, a sequence containing <U+0EC8 LAO TONE MAI EK, U+0ECD LAO NIGGAHITA> should be displayed with the *niggahita* above the *mai ek*, and a sequence containing <U+0ECD LAO NIGGAHITA, U+0EC8 LAO TONE MAI EK> should be displayed with the *mai ek* above the *niggahita*.

This does not preclude input processors from helping the user by pointing out or correcting typing mistakes, perhaps taking into account the language. For example, because the string <*mai ek, niggahita*> is not useful for the Lao language and is likely a typing mistake, an input processor could reject it or correct it to <*niggahita, mai ek*>.

When the character U+0EB3 LAO VOWEL SIGN AM follows one or more tone marks (U+0EC8..U+0ECB), the *niggahita* that is part of the *sara am* should be displayed below those tone marks. In particular, a sequence containing <U+0EC8 LAO TONE

MAI EK, U+0EB3 LAO VOWEL SIGN AM> should be displayed with the *mai ek* above the *niggahita*.

Spacing. Lao words are not separated by spaces. Use of spaces and Latin-based punctuation is very similar to that of Thai. For more information, as well as the use of U+200B ZERO WIDTH SPACE for explicit word break or line break opportunities, see Thai *Spacing*.

Lao Aspirated Nasals. The Unicode character encoding includes two ligatures for Lao: U+0EDC LAO HO NO and U+0EDD LAO HO MO. They correspond to sequences of [h] plus [n] or [h] plus [m] without ligating. Their function in Lao is to provide versions of the [n] and [m] consonants with a different inherent tonal implication.

Transcription of Pali and Sanskrit. Traditionally the Lao script is not used to write Pali and Sanskrit. The Lao consonant repertoire originally contained only the letters needed by the modern Lao language. An extended writing system was designed in the 1930s by Maha Sila Viravong to transcribe consonant clusters and additional consonants of Pali. The additional characters required by the extension are listed in *Table 16-3*.

Table 16-3. Additional Characters for Pali and Sanskrit

Characters	Note
0EBA	Virama
0E86, 0E89, 0E8C, 0E8E..0E93, 0E98, 0EA0, 0EAC	Consonant letters
0EA8..0EA9	Sanskrit-specific consonant letters

U+0EBA LAO SIGN PALI VIRAMA marks the removal of the inherent vowel of a consonant letter, and does not indicate conjoining or stacking behavior. U+0EA8 LAO LETTER SANSKRIT SHA and U+0EA9 LAO LETTER SANSKRIT SSA are used only in Sanskrit.

Implementations should not assume transliteration mappings or a cognate relationship between all Lao and Thai characters based on their relative locations in the blocks. For example, Pali *nya*, a cognate of U+0E0D THAI CHARACTER YO YING, is encoded at U+0E8E instead of the corresponding location U+0E8D because the latter is already occupied by Lao *nyo*, a phonetically related non-cognate of Thai *yo ying*.

16.3 Myanmar

16.3.1 Myanmar: U+1000–U+109F

The Myanmar script is used to write Burmese, the majority language of Myanmar (formerly called Burma). Variations and extensions of the script are used to write other languages of the region, such as Mon, Karen, Kayah, Shan, and Palaung, as well as Pali and Sanskrit. The Myanmar script was formerly known as the Burmese script, but the term "Myanmar" is now preferred.

The Myanmar writing system derives from a Brahmi-related script borrowed from South India in about the eighth century to write the Mon language. The first inscription in the Myanmar script dates from the eleventh century and uses an alphabet almost identical to that of the Mon inscriptions. Aside from rounding of the originally square characters, this script has remained largely unchanged to the present. It is said that the rounder forms were developed to permit writing on palm leaves without tearing the writing surface of the leaf.

The Myanmar script shares structural features with other Brahmi-based scripts such as Khmer: consonant symbols include an inherent "a" vowel; various signs are attached to a consonant to indicate a different vowel; medial consonants are attached to the consonant; and the overall writing direction is from left to right.

Standards. There is not yet an official national standard for the encoding of Myanmar/Burmese. The current encoding was prepared with the consultation of experts from the Myanmar Information Technology Standardization Committee (MITSC) in Yangon (Rangoon). The MITSC, formed by the government in 1997, consists of experts from the Myanmar Computer Scientists' Association, Myanmar Language Commission, and Myanmar Historical Commission.

Encoding Principles. As with Indic scripts, the Myanmar encoding represents only the basic underlying characters; multiple glyphs and rendering transformations are required to assemble the final visual form for each syllable. Characters and combinations that may appear visually identical in some fonts, such as U+101D ○ MYANMAR LETTER WA and U+1040 ○ MYANMAR DIGIT ZERO, are distinguished by their underlying encoding.

Composite Characters. As is the case in many other scripts, some Myanmar letters or signs may be analyzed as composites of two or more other characters and are not encoded separately. The following are three examples of Myanmar letters represented by combining character sequences:

U+1000 က *ka* + U+1031 ◌ေ *vowel sign e* + U+102C ◌ာ *vowel sign aa* → ကော /kàw/

U+1000 ၐ *ka* + U+1031 ေ *vowel sign e* + U+102C ာ *vowel sign aa* + U+103A ် *asat* → ေကာ် */kaw/*

U+1000 ၐ *ka* + U+102D ိ *vowel sign i* + U+102F ု *vowel sign u* → ကို */ko/*

Encoding Subranges. The basic consonants, medials, independent vowels, and dependent vowel signs required for writing the Myanmar language are encoded at the beginning of the Myanmar block. Those are followed by script-specific digits, punctuation, and various signs. The last part of the block contains extensions for consonants, medials, vowels, and tone marks needed to represent historic text and various other languages. These extensions support Pali and Sanskrit, as well as letters and tone marks for Mon, Karen, Kayah, and Shan. The extensions include two tone marks for Khamti Shan and two vowel signs for Aiton and Phake, but the majority of the additional characters needed to support those languages are found in the Myanmar Extended-A block.

Conjuncts. As in other Indic-derived scripts, conjunction of two consonant letters is indicated by the insertion of a virama U+1039 ◌ MYANMAR SIGN VIRAMA between them. It causes the second consonant to be displayed in a smaller form below the first; the virama is not visibly rendered.

Kinzi. The conjunct form of U+1004 င MYANMAR LETTER NGA is rendered as a superscript sign called *kinzi*. That superscript sign is not encoded as a separate mark, but instead is simply the rendering form of the *nga* in a conjunct context. The *nga* is represented in logical order first in the sequence, before the consonant which actually bears the visible *kinzi* superscript sign in final rendered form. For example, *kinzi* applied to U+1000 ၐ MYANMAR LETTER KA would be written via the following sequence:

U+1004 င *nga* + U+103A ် *asat* + U+1039 ◌ *virama* + U+1000 ၐ *ka* → ကိ *ka*

Note that this sequence includes both U+103A *asat* and U+1039 *virama* between the *nga* and the *ka*. Use of the *virama* alone would ordinarily indicate stacking of the consonants, with a small *ka* appearing under the *nga*. Use of the *asat* killer in addition to the *virama* gives a sequence that can be distinguished from normal stacking: the sequence <U+1004, U+103A, U+1039> always maps unambiguously to a visible *kinzi* superscript sign on the following consonant.

Medial Consonants. The Myanmar script traditionally distinguishes a set of "medial" consonants: forms of *ya*, *ra*, *wa*, and *ha* that are considered to be modifiers of the syllable's vowel. Graphically, these medial consonants are sometimes written as subscripts, but sometimes, as in the case of *ra*, they surround the base consonant instead. In the Myanmar encoding, the medial consonants are encoded separately. For example, the word ေကျွ [kjwei] ("to drop off") would be written via the following sequence:

U+1000 က *ka* + U+103C ◌ *medial ra* + U+103D ◌ *medial wa* + U+1031 ◌ *vowel sign e* → ကြွေ */kjwei/*

In Pali and Sanskrit texts written in the Myanmar script, as well as in older orthographies of Burmese, the consonants *ya*, *ra*, *wa*, and *ha* are sometimes rendered in subjoined form. In those cases, U+1039 ◌ MYANMAR SIGN VIRAMA and the regular form of the consonant are used.

Asat. The *asat*, or *killer*, is a visibly displayed sign. In some cases it indicates that the inherent vowel sound of a consonant letter is suppressed. In other cases it combines with other characters to form a vowel letter. Regardless of its function, this visible sign is always represented by the character U+103A ◌ MYANMAR SIGN ASAT.

Contractions. In a few Myanmar words, the repetition of a consonant sound is written with a single occurrence of the letter for the consonant sound together with an *asat* sign. This *asat* sign occurs immediately after the double-acting consonant in the coded representation:

U+101A ယ *ya* + U+1031 ◌ *vowel sign e* + U+102C ◌ *vowel sign aa* + U+1000 က *ka* + U+103A ◌ *asat* + U+103B ◌ *medial ya* + U+102C ◌ *vowel sign aa* + U+1038 ◌ *visarga* → ယောက်ျား man, husband

U+1000 က *ka* + U+103B ◌ *medial ya* + U+103D ◌ *medial wa* + U+1014 န *na* + U+103A ◌ *asat* + U+102F ◌ *vowel sign u* + U+1015 ပ *pa* + U+103A ◌ *asat* → ကျွန်ုပ် I (first person singular)

Great sa. The *great sa* is encoded as U+103F ဿ MYANMAR LETTER GREAT SA. This letter should be represented with <U+103F>, while the sequence <U+101E, U+1039, U+101E> should be used for the regular conjunct form of two *sa*, ssa, and the sequence <U+101E, U+103A, U+101E> should be used for the form with an *asat sign*, သ်သ.

Tall aa. The two shapes ◌ါ and ◌ာ are both used to write the sound /a/. In Burmese orthography, both shapes are used, depending on the visual context. In S'gaw Karen orthography, only the tall form is used. For this reason, two characters are encoded: U+102B ◌ါ MYANMAR VOWEL SIGN TALL AA and U+102C ◌ာ MYANMAR VOWEL SIGN AA. In Burmese texts, the coded character appropriate to the visual context should be used.

Ordering of Syllable Components. Dependent vowels and other signs are encoded after the consonant to which they apply, except for *kinzi*, which precedes the consonant. Characters occur in the relative order shown in *Table 16-4*.

Table 16-4. Modern Burmese Syllabic Structure

Class	Example	Encoding

kinzi	⟨	<U+1004, U+103A, U+1039>
consonant and vowel letters	က	[U+1000..U+1021, U+1023..U+1027, U+1029, U+102A, U+103F, U+104E]
subscript consonant	⟨	<U+1039, [U+1000..U+1008, U+100A..U+1019, U+101B, U+101C, U+101E, U+1020, U+1021]>
asat sign	⟨	U+103A
medial ya (potentially followed by asat sign)	⟨	<U+103B, (U+103A)>
medial ra	⟨	U+103C
medial wa	⟨	U+103D
medial ha	⟨	U+103E
vowel sign e	ေ	U+1031
vowel sign i, ii, ai	⟨, ⟨, ⟨	[U+102D, U+102E, U+1032]
vowel sign u, uu	⟨, ⟨	[U+102F, U+1030]
vowel sign tall aa, aa (potentially followed by asat sign)	⟨, ⟨	<[U+102B, U+102C], (U+103A)>
anusvara	⟨	U+1036
dot below	⟨	U+1037
visarga	⟨	U+1038

U+1031 ေ MYANMAR VOWEL SIGN E is encoded after its consonant (as in the earlier example), although in visual presentation its glyph appears before (to the left of) the consonant form.

Table 16-4 nominally refers to the character sequences used in representing the syllabic structure of the modern Burmese language proper. Canonical normalization may result in a different ordering, specifically with some occurrences of U+103A ⟨ MYANMAR SIGN ASAT reordered after U+1037 ⟨ MYANMAR SIGN DOT BELOW. As such reorderings are canonically equivalent, implementations should support both orders and treat them as fundamentally the same text.

Table 16-4 would require further extensions and modifications to cover various other languages, such as Karen, Mon, Shan, Sanskrit, and Old Burmese, which also use the Myanmar script. For some such extensions and modifications, refer to Unicode Technical Note #11, "Representing Myanmar in Unicode: Details and Examples," or also Microsoft Typography's "Creating and Supporting OpenType Fonts for Myanmar Script." Note that those documents are not normative for the Unicode Standard, and they also differ from each other in some details.

Spacing. Myanmar does not use any whitespace between words. If explicit word break or line break opportunities are desired—for example, for the use of automatic line layout algorithms—the character U+200B ZERO WIDTH SPACE should be used to place invisible marks for such breaks. The ZERO WIDTH SPACE can grow to have a

visible width when justified. Spaces are used to mark phrases. Some phrases are relatively short (two or three syllables).

16.3.2 Myanmar Extended-A: U+AA60–U+AA7F

This block provides additional characters to support Khamti Shan, Aiton and Phake. The block also contains a few additional tone marks for Pa'o Karen and Tai Laing, and two additional letters for Shwe Palaung. Khamti Shan is spoken by approximately 14,000 people in Myanmar and India. Aiton and Phake are smaller language communities of around 2,000 each. Many of the characters needed for these languages are provided by the main Myanmar block. Khamti Shan, Aiton, and Phake writing conventions are based on Shan, and as such follow the general Myanmar model of encoding.

16.3.3 Khamti Shan

The Khamti Shan language has a long literary tradition which has largely been lost, for a variety of reasons. The old script did not mark tones, and it had a scribal tradition that encouraged restriction to a reading elite whose traditions have not been passed on. The script has recently undergone a revival, with plans for it to be taught throughout the Khamti-Shan-speaking regions in Myanmar. A new version of the script has been adopted by the Khamti in Myanmar. The Khamti Shan characters in the Myanmar Extended-A block supplement those in the Myanmar block and provide complete support for the modern Khamti Shan writing system as written in Myanmar. Another revision of the old script was made in India under the leadership of Chau Khouk Manpoong in the 1990s. That revision has not gained significant popularity, although it enjoys some currency today.

Consonants. Approximately half of the consonants used in Khamti Shan are encoded in the Myanmar block. Following the conventions used for Shan, Mon, and other extensions to the Myanmar script, separate consonants are encoded specifically for Khamti Shan in this block when they differ significantly in shape from corresponding letters conveying the same consonant sounds in Myanmar proper. Khamti Shan also uses the three Myanmar medial consonants encoded in the range U+101B..U+101D.

The consonants in this block are displayed in the code charts using a Burmese style, so that glyphs for the entire Myanmar script are harmonized in a single typeface. However, the local style preferred for Khamti Shan is slightly different, typically adding a small dot to each character.

Vowels. The vowels and dependent vowel signs used in Khamti Shan are located in the Myanmar block.

Tones. Khamti Shan has eight tones. Seven of these are written with explicit tone marks; one is unmarked. All of the explicit tone marks are encoded in the Myanmar block. Khamti Shan makes use of four of the Shan tone marks and the *visarga*. In addition, two Khamti Shan-specific tone marks are separately encoded. These tone marks for Khamti Shan are listed in *Table 16-5*.

Table 16-5. Khamti Shan Tone Marks

Tone	Character
1	U+109A MYANMAR SIGN KHAMTI TONE-1
2	U+1089 MYANMAR SIGN SHAN TONE-5
3	U+109B MYANMAR SIGN KHAMTI TONE-3
4	U+1087 MYANMAR SIGN SHAN TONE-2
5	U+1088 MYANMAR SIGN SHAN TONE-3
6	U+1038 MYANMAR SIGN VISARGA
7	*unmarked*
8	U+108A MYANMAR SIGN SHAN TONE-6

The vertical positioning of the small circle in some of these tone marks is considered distinctive. U+109A MYANMAR SIGN KHAMTI TONE-1 (with a high position) is not the same as U+108B MYANMAR SIGN SHAN COUNCIL TONE-2 (with a mid-level position). Neither of those should be confused with U+1089 MYANMAR SIGN SHAN TONE-5 (with a low position).

The tone mark characters in Shan fonts are typically displayed with open circles. However, in Khamti Shan, the circles in the tone marks normally are filled in (black).

Digits. Khamti Shan uses the Shan digits from the range U+1090..U+109A.

Other Symbols. Khamti Shan uses the punctuation marks U+104A MYANMAR SIGN LITTLE SECTION and U+104B MYANMAR SIGN SECTION. The repetition mark U+AA70 MYANMAR MODIFIER LETTER KHAMTI REDUPLICATION is functionally equivalent to U+0E46 THAI CHARACTER MAIYAMOK.

Three logogram characters are also used. These logograms can take tone marks, and their meaning varies according to the tone they take. They are used when transcribing speech rather than in formal writing. For example, U+AA75 MYANMAR LOGOGRAM KHAMTI QN takes three tones and means "negative," "giving" or "yes," according to which tone is applied. The other two logograms are U+AA74 MYANMAR LOGOGRAM KHAMTI OAY and U+AA76 MYANMAR LOGOGRAM KHAMTI HM.

Subjoined Characters. Khamti Shan does not use subjoined characters.

Historical Khamti Shan. The characters of historical Khamti Shan are for the most part identical to those used in the New Khamti Shan orthography. Most variation is

merely stylistic. There were no Pali characters. The only significant character difference lies with *ra*—which follows Aiton and Phake in using a *la* with *medial ra* (U+AA7A MYANMAR LETTER AITON RA).

During the development of the New Khamti Shan orthography a few new character shapes were introduced that were subsequently revised. Because materials have been published using these shapes, and these shapes cannot be considered stylistic variants of other characters, these characters are separately encoded in the range U+AA71..U+AA73.

16.3.4 Aiton and Phake

The Aiton and Phake writing systems are very closely related. There are a small number of differences in shape between Aiton and Phake characters, but these are considered only glyphic differences. As for Khamti Shan, most of the characters needed for Aiton and Phake are found in the Myanmar block.

Consonants. U+107A MYANMAR LETTER SHAN NYA is used rather than following the Khamti U+AA65 MYANMAR LETTER KHAMTI NYA because the character shape follows Shan rather than Khamti.

Subjoined Consonants. Aiton and Phake have a system of subjoining consonants to chain syllables in a polysyllabic word. This system follows that of Burmese and is encoded in the same way: with U+1039 MYANMAR SIGN VIRAMA followed by the code of the consonant being subjoined. The following characters may take a subjoined form, which takes the same shape as the base character but smaller: U+1000, U+AA61, U+1010, U+1011, U+1015, U+101A, U+101C. No other subjoined characters are known in Aiton and Phake.

Vowels. The vowels follow Shan for the most part, and are therefore based on the characters in the Myanmar block. In addition to the simple vowels there are a number of diphthongs in Aiton and Phake. One vowel and one diphthong required for these languages were added as extensions at the end of the Myanmar block. A number of the vowel letters and diphthongs in the Aiton and Phake alphabets are composed of a sequence of code points. For example, the vowel *-ue* is represented by the sequence <U+102D, U+102F, U+101D, U+103A>.

Ligatures. The characters in the range U+AA77..U+AA79 are a set of ligature symbols that follow the same principles used for U+109E MYANMAR SYMBOL SHAN ONE and U+109F MYANMAR SYMBOL SHAN EXCLAMATION. They are symbols that constitute a word in their own right and do not take diacritics.

Tones. Traditionally tones are not marked in Aiton and Phake, although U+109C MYANMAR VOWEL SIGN AITON A (*short -a*) can be used as a type of tone marker. All

proposed patterns for adding tone marking to Aiton and Phake can be represented with the tone marks used for Shan or Khamti Shan.

16.3.5 Myanmar Extended-B: U+A9E0–U+A9FF

This block contains additional characters for Shan Pali that represent Sanskrit sounds written in Shan. It also contains many characters for Tai Laing, a Tai language related to Khamti and spoken in the Kachin state of Myanmar.

Digits. Tai Laing has a distinct set of digits that differ in appearance from both the main set of Myanmar digits and the Shan digits encoded in the main Myanmar block.

16.3.6 Myanmar Extended-C: U+116D0–U+116FF

This block contains two additional sets of digits, one for the Pa'O language, and one for Eastern Pwo Karen. The writers of these languages often use standard Myanmar digits or the Western (ASCII) digits, but use of the language-specific forms of digits is also found.

16.4 Khmer

16.4.1 Khmer: U+1780–U+17FF

Khmer, also known as Cambodian, is the official language of the Kingdom of Cambodia. Mutually intelligible dialects are also spoken in northeastern Thailand and in the Mekong Delta region of Vietnam. Although Khmer is not an Indo-European language, it has borrowed much vocabulary from Sanskrit and Pali, and religious texts in those languages have been both transliterated and translated into Khmer. The Khmer script is also used to render a number of regional minority languages, such as Tampuan, Krung, and Cham.

The Khmer script, called *aksaa khmae* ("Khmer letters"), is also the official script of Cambodia. It is descended from the Brahmi script of South India, as are Thai, Lao, Myanmar, Old Mon, and others. The exact sources have not been determined, but there is a great similarity between the earliest inscriptions in the region and the Pallawa script of the Coromandel coast of India. Khmer has been a unique and independent script for more than 1,400 years. Modern Khmer has two basic styles of script: the *aksaa crieng* ("slanted script") and the *aksaa muul* ("round script"). There is no fundamental structural difference between the two. The slanted script (in its "standing" variant) is chosen as representative in the code charts.

16.4.2 Principles of the Khmer Script

Structurally, the Khmer script has many features in common with other Brahmi-derived scripts, such as Devanagari and Myanmar. Consonant characters bear an inherent vowel sound, with additional signs placed before, above, below, and/or after the consonants to indicate a vowel other than the inherent one. The overall writing direction is left to right.

In comparison with the Devanagari script, explained in detail in *Section 12.1, Devanagari*, the Khmer script has developed several distinctive features during its evolution.

Glottal Consonant. The Khmer script has a consonant character for a glottal stop (*qa*) that bears an inherent vowel sound and can have an optional vowel sign. While Khmer also has independent vowel characters like Devanagari, as shown in *Table 16-6*, in principle many of its sounds can be represented by using *qa* and a vowel sign. This does not mean these representations are always interchangeable in real words. Some words are written with one variant to the exclusion of others.

Table 16-6. Independent Khmer Vowel Characters

Name	Independent Vowel	Qa with Vowel Sign(s)

Name	Independent Vowel	Ra or La with Vowel Sign
i	ឥ	អិ, អ៊ិ, អ៊ី
ii	ឦ	អី, អ៊ិ
u	ឧ	អុ, អ៊ុ
uk	ឩ	អុក
uu	ឩ	អូ, អ៊ូ
uuv	ឩ	អូវ
e	ឯ	អេ, អែ
ai	ឰ	អៃ
oo	ឱ, ឲ	អោ
au	ឳ	អៅ
Name	**Independent Vowel**	**Ra or La with Vowel Sign**
ry	ឫ	រឹ
ryy	ឬ	រឺ
ly	ឭ	លឹ
lyy	ឮ	លឺ

Subscript Consonants. Subscript consonant signs differ from independent consonant characters and are called *coeng* (literally, "foot, leg") after their subscript position. While a consonant character can constitute an orthographic syllable by itself, a subscript consonant sign cannot. Note that U+17A1 ឡ KHMER LETTER LA does not have a corresponding subscript consonant sign in standard Khmer, but does have a subscript in the Khmer script used in Thailand.

Subscript consonant signs are used to represent any consonant following the first consonant in an orthographic syllable. They also have an inherent vowel sound, which may be suppressed if the syllable bears a vowel sign or another subscript consonant.

The subscript consonant signs are often used to represent a consonant cluster. Two consecutive consonant characters cannot represent a consonant cluster because the inherent vowel sound in between is retained. To suppress the vowel, a subscript consonant sign (or rarely a subscript independent vowel) replaces the second consonant character. Theoretically, any consonant cluster composed of any number of consonant sounds without inherent vowel sounds in between can be represented systematically by a consonant character and as many subscript consonant signs as necessary.

Examples of subscript consonant signs for a consonant cluster follow:

ល្ង *lo + coeng + ngo* [lŋɔː] "sesame" (compare លង *lo + ngo* [lɔːŋ] "to haunt")

លក្ស្មី *lo + ka + coeng + sa + coeng + mo + ii* [lɛəksmei] "beauty, luck"

កាហ្វេ *ka + aa + ha + coeng + vo + e* [kaːfeː] "coffee"

The subscript consonant signs in the Khmer script can be used to denote a final consonant, although this practice is uncommon.

Examples of subscript consonant signs for a closing consonant follow:

ទាំ្ង *to + aa + nikahit + coeng + ngo* [tɛəŋ] "both" (= ទាំង) (≠ *ទ្ង [tŋɔəm])

ហើ្យ *ha + oe + coeng + yo* [haəi] "already" (= ហើយ) (≠ *ហ្ញ [hjaə])

While these subscript consonant signs are usually attached to a consonant character, they can also be attached to an independent vowel character. Although this practice is relatively rare, it is used in one very common word, meaning "to give."

Examples of subscript consonant signs attached to an independent vowel character follow:

ឱ្យ *qoo-1 + coeng + yo* [ʔaoi] "to give" (= ឱយ and also ឲ្យ)

ឱ្ម *qoo-1 + coeng + mo* [ʔaom] "exclamation of solemn affirmation" (= ឱម)

Subscript Independent Vowel Signs. Some independent vowel characters also have corresponding subscript independent vowel signs, although these are rarely used today.

Examples of subscript independent vowel signs follow:

ផ្ឯម *pha + coeng + qe + mo* [pʰʔaem] "sweet" (= ផ្ឰម *pha + coeng + qa + ae + mo*)

ហ្ឫទ៏ឩ *ha + coeng + ry + to + samyok sannya + yo* [harutej] "heart" (*royal*) (= ហឫទ៏ឩ *ha + ry + to + samyok sannya + yo*)

Consonant Registers. The Khmer language has a richer set of vowels than the languages for which the ancestral script was used, although it has a smaller set of consonant sounds. The Khmer script takes advantage of this situation by assigning different characters to represent the same consonant using different inherent vowels. Khmer consonant characters and signs are organized into two series or registers, whose inherent vowels are nominally *-a* in the first register and *-o* in the second register, as shown in <u>Table 16-7</u>.

Table 16-7. Two Registers of Khmer Consonants

Row	First Register	Second Register
1	ក *ka* [kɔː] "neck"	គ *ko* [kɔ̀ː] "mute"
2	រ៉ *ro + muusikatoan* [rɔː] "small saw"	រ *ro* [rɔ̀ː] "fence (in the water)"
3	សក *sa + ka* [sɔːk] "to peel, to shed one's skin"	ស៊ក *sa + triisap + ka* [sɔ̀ːk] "to insert"
4	បក *ba + ka* [bɔːk] "to return"	*ប៊ក *ba + triisap + ka* [bɔ̀ːk]
5	ប៉ម *ba + muusikatoan + mo* [pɔːm] "blockhouse"	ពម *po + mo* [pɔ̀ːm] "to put into the mouth"
6	ក្ងុរ *ka + u + ro* [koː] "to stir"	គុរ *ko + u + ro* [kuː] "to sketch"

The register of a consonant character is generally reflected on the last letter of its transliterated name. Some consonant characters and signs have a counterpart whose consonant sound is the same but whose register is different, as *ka* and *ko* in the first row of the table. For the other consonant characters and signs, two "shifter" signs are available. U+17C9 ◌̊ KHMER SIGN MUUSIKATOAN converts a consonant character and sign from the second to the first register, while U+17CA ◌̃ KHMER SIGN TRIISAP converts a consonant from the first register to the second (rows 2–4). To represent *pa*, however, *muusikatoan* is attached not to *po* but to *ba*, in an exceptional use (row 5). The phonetic value of a dependent vowel sign may also change depending on the context of the consonant(s) to which it is attached (row 6).

Encoding Principles. Like other related scripts, the Khmer encoding represents only the basic underlying characters; multiple glyphs and rendering transformations are required to assemble the final visual form for each orthographic syllable. Individual characters, such as U+1789 ញ KHMER LETTER NYO, may assume variant forms depending on the other characters with which they combine.

Subscript Consonant Signs. In the way that many Cambodians analyze Khmer today, subscript consonant signs are considered to be different entities from consonant characters. The Unicode Standard does not assign independent code points for the subscript consonant signs. Instead, each of these signs is represented by the sequence of two characters: a special control character (U+17D2 ◌្ KHMER SIGN

COENG) and a corresponding consonant character. This is analogous to the virama model employed for representing conjuncts in other related scripts. Subscripted independent vowels are encoded in the same manner. Because the *coeng sign* character does not exist as a letter or sign in the Khmer script, the Unicode model departs from the ordinary way that Khmer is conceived of and taught to native Khmer speakers. Consequently, the encoding may not be intuitive to a native user of the Khmer writing system, although it is able to represent Khmer correctly.

U+17D2 ◌ KHMER SIGN COENG is not actually a *coeng* but a *coeng* generator, because *coeng* in Khmer refers to the subscript consonant sign. The glyph for KHMER SIGN COENG shown in the code charts is arbitrary and is not actually rendered directly; the dotted box around the glyph indicates that special rendering is required. To aid Khmer script users, a listing of typical Khmer subscript consonant letters has been provided in *Table 16-8* together with their descriptive names following preferred Khmer practice. While the Unicode encoding represents both the subscripts and the combined vowel letters with a pair of code points, they should be treated as a unit for most processing purposes. In other words, the sequence functions as if it had been encoded as a single character. A number of independent vowels also have subscript forms, as shown in *Table 16-8*.

Table 16-8. Khmer Subscript Consonant Signs

Glyph	Code	Name
◌	17D2 1780	*khmer consonant sign coeng ka*
◌	17D2 1781	*khmer consonant sign coeng kha*
◌	17D2 1782	*khmer consonant sign coeng ko*
◌	17D2 1783	*khmer consonant sign coeng kho*
◌	17D2 1784	*khmer consonant sign coeng ngo*
◌	17D2 1785	*khmer consonant sign coeng ca*
◌	17D2 1786	*khmer consonant sign coeng cha*
◌	17D2 1787	*khmer consonant sign coeng co*
◌	17D2 1788	*khmer consonant sign coeng cho*
◌	17D2 1789	*khmer consonant sign coeng nyo*
◌	17D2 178A	*khmer consonant sign coeng da*
◌	17D2 178B	*khmer consonant sign coeng ttha*
◌	17D2 178C	*khmer consonant sign coeng do*

ស្ឋ	17D2 178D	*khmer consonant sign coeng ttho*
ណ	17D2 178E	*khmer consonant sign coeng na*
ត	17D2 178F	*khmer consonant sign coeng ta*
ថ	17D2 1790	*khmer consonant sign coeng tha*
ទ	17D2 1791	*khmer consonant sign coeng to*
ធ	17D2 1792	*khmer consonant sign coeng tho*
ន	17D2 1793	*khmer consonant sign coeng no*
ប	17D2 1794	*khmer consonant sign coeng ba*
ផ	17D2 1795	*khmer consonant sign coeng pha*
ព	17D2 1796	*khmer consonant sign coeng po*
ភ	17D2 1797	*khmer consonant sign coeng pho*
ម	17D2 1798	*khmer consonant sign coeng mo*
យ	17D2 1799	*khmer consonant sign coeng yo*
រ	17D2 179A	*khmer consonant sign coeng ro*
ល	17D2 179B	*khmer consonant sign coeng lo*
វ	17D2 179C	*khmer consonant sign coeng vo*
ឝ	17D2 179D	*khmer consonant sign coeng sha*
ឞ	17D2 179E	*khmer consonant sign coeng ssa*
ស	17D2 179F	*khmer consonant sign coeng sa*
ហ	17D2 17A0	*khmer consonant sign coeng ha*
ឡ	17D2 17A1	*khmer consonant sign coeng la*
អ	17D2 17A2	*khmer vowel sign coeng qa*

As noted earlier, <U+17D2, U+17A1> represents a subscript form of *la* that is not used in Cambodia, although it is employed in Thailand.

Dependent Vowel Signs. Most of the Khmer dependent vowel signs are represented with a single character that is applied after the base consonant character and optional subscript consonant signs. Three of these Khmer vowel signs are not encoded as

single characters in in the Unicode Standard. The vowel sign *am* is encoded as a nasalization sign, U+17C6 ◌̊ KHMER SIGN NIKAHIT. Two vowel signs, *om* and *aam*, have not been assigned independent code points. They are represented by the sequence of a vowel (U+17BB ◌ KHMER VOWEL SIGN U and U+17B6 ◌ា KHMER VOWEL SIGN AA, respectively) and U+17C6 ◌̊ KHMER SIGN NIKAHIT.

The *nikahit* is superficially similar to *anusvara*, the nasalization sign in the Devanagari script, although in Khmer it is usually regarded as a vowel sign *am*. *Anusvara* not only represents a special nasal sound, but also can be used in place of one of the five nasal consonants homorganic to the subsequent consonant (velar, palatal, retroflex, dental, or labial, respectively). *Anusvara* can be used concurrently with any vowel sign in the same orthographic syllable. *Nikahit*, in contrast, functions differently. Its final sound is [m], irrespective of the type of the subsequent consonant. It is not used concurrently with the vowels *ii*, *e*, *ua*, *oe*, *oo*, and so on, although it is used with the vowel signs *aa* and *u*. In these cases the combination is sometimes regarded as a unit—*aam* and *om*, respectively. The sound that *aam* represents is [ɔəm], not [aːm]. The sequences used for these combinations are shown in *Table 16-9*.

Table 16-9. Khmer Composite Dependent Vowel Signs with Nikahit

Glyph	Code	Name
◌̊	17BB 17C6	*khmer vowel sign om*
◌ាំ	17B6 17C6	*khmer vowel sign aam*

Examples of dependent vowel signs ending with [m] follow:

ដុំ *da + nikahit* [dɔm] "to pound" (compare ដុម *da + mo* [dɔːm] "nectar")

ពាំ *po + aa + nikahit* [pɔəm] "to carry in the beak" (compare ពាម *po + aa + mo* [pèəm] "mouth of a river")

Independent Vowel Characters. In Khmer, as in other Brahmic scripts, some independent vowels have their own letterforms, although the sounds they represent may more often be represented with the consonant character for the glottal stop (U+17A2 អ KHMER LETTER QA) modified by vowel signs (and optionally a consonant character). These independent vowels are encoded as separate characters in the Unicode Standard.

Subscript Independent Vowel Signs. Some independent vowels have corresponding subscript independent vowel signs, although these are rarely used. Each is represented by the sequence of U+17D2 ◌្ KHMER SIGN COENG and an independent vowel, as shown in *Table 16-10*.

Table 16-10. Khmer Subscript Independent Vowel Signs

Glyph	Code	Name
◌	17D2 17A7	*khmer independent vowel sign coeng qu*
◌	17D2 17AB	*khmer independent vowel sign coeng ry*
◌	17D2 17AC	*khmer independent vowel sign coeng ryy*
◌	17D2 17AF	*khmer independent vowel sign coeng qe*

Other Signs as Syllabic Components. The Khmer sign *robat* historically corresponds to the Devanagari *repha*, a representation of syllable-initial *r-*. However, the Khmer script can treat the initial *r-* in the same way as the other initial consonants—namely, a consonant character *ro* and as many subscript consonant signs as necessary. Some old loan words from Sanskrit and Pali include *robat*, but in some of them the *robat* is not pronounced and is preserved in a fossilized spelling. Because *robat* is a distinct sign from the consonant character *ro*, the Unicode Standard encodes U+17CC ◌̃ KHMER SIGN ROBAT, but it treats the Devanagari *repha* as a part of a ligature when encoding it. The authoritative Chuon Nath dictionary sorts *robat* as if it were a base consonant character, just as the *repha* is sorted in scripts that use it. The consonant over which *robat* resides is then sorted as if it were a subscript.

Examples of consonant clusters beginning with *ro* and *robat* follow:

រាជឫសី *ro + aa + co + ro + coeng + sa + ii* [rèəcrsei] "king hermit"

អារ្យ *qa + aa + yo + robat* [ʔaːrja] "civilized" (= អារ្យ *qa + aa + ro + coeng + yo*)

ពត៌មាន *po + ta + robat + mo + aa + no* [pɔːdɔmèən] "news" (compare Sanskrit वर्तमान *vartamāna* "the present time")

U+17DD ◌̃ KHMER SIGN ATTHACAN is a rarely used sign that denotes that the base consonant character keeps its inherent vowel sound. This use contrasts with U+17D1 ◌̄ KHMER SIGN VIRIAM, which indicates the removal of the inherent vowel sound of a base consonant. U+17CB ◌́ KHMER SIGN BANTOC shortens the vowel sound of the previous orthographic syllable. U+17C7 ◌ះ KHMER SIGN REAHMUK, U+17C8 ◌: KHMER SIGN YUUKALEAPINTU, U+17CD ◌̆ KHMER SIGN TOANDAKHIAT, U+17CE ◌̟ KHMER SIGN KAKABAT, U+17CF ◌̋ KHMER SIGN AHSDA, and U+17D0 ◌̌ KHMER SIGN SAMYOK SANNYA are also explicitly encoded signs used to compose an orthographic syllable.

Ligatures. Some vowel signs form ligatures with consonant characters and signs. These ligatures are not encoded separately, but should be presented graphically by the rendering software. Some common ligatures are shown in <u>*Figure 16-1*</u>.

Figure 16-1. Common Ligatures in Khmer

ក *ka* + ា *aa* + រ *ro* = ការ [kaː] "job"

ប *ba* + ា *aa* = បា [baː] "father, male of an animal"; used to prevent confusion with ហ *ha*

ប *ba* + ៅ *au* = បៅ [baw] "to suck"

ម *mo* + ្ស *coeng sa* + ៅ *au* = ម្សៅ [msaw] "powder"

ស *sa* + ង *ngo* + ្ខ *coeng kha* + ្យ *coeng yo* + ា *aa* = សង្ខ្យា [sɔŋkʰjaː] "counting"

Multiple Glyphs. A single character may assume different forms according to context. For example, a part of the glyph for *nyo* is omitted when a subscript consonant sign is attached. The implementation must render the correct glyph according to context. *Coeng nyo* also changes its shape when it is attached to *nyo*. The correct glyph for the sequence <U+17D2 ◌្ KHMER SIGN COENG, U+1789 ញ KHMER LETTER NYO> is rendered according to context, as shown in *Figure 16-2*. This kind of glyph alternation is very common in Khmer. Some spacing subscript consonant signs change their height depending on the orthographic context. Similarly, the vertical position of many signs varies according to context. Their presentation is left to the rendering software.

U+17B2 ឲ KHMER INDEPENDENT VOWEL QOO TYPE TWO is thought to be a variant of U+17B1 ឱ KHMER INDEPENDENT VOWEL QOO TYPE ONE, but it is explicitly encoded in the Unicode Standard. The variant is used in very few words, but these include the very common word *aoi* "to give," as noted in *Figure 16-2*.

Figure 16-2. Common Multiple Forms in Khmer

ញញឹម *nyo* + *nyo* + *y* + *mo* [ɲɔ̀ɲum] "to smile"

ចិញ្ចើម *ca* + *i* + *nyo* + *coeng* + *ca* + *oe* + *mo* [ceɲcaəm] "eyebrow"

ស្ងប់ *sa* + *coeng nyo* + *ba* + *bantoc* [sɲɔp] "to respect"

កញ្ញា *ka* + *nyo* + *coeng* + *nyo* + *aa* [kaɲɲaː] "girl, Miss, September"

ឲ្យ *qoo-2* + *coeng* + *yo* (= ឱ្យ *qoo-1* + *coeng* + *yo*) [ʔaoi] "to give"

Characters Whose Use Is Discouraged. Some of the Khmer characters encoded in the Unicode Standard are not recommended for use for various reasons.

U+17A3 KHMER INDEPENDENT VOWEL QAQ and U+17A4 KHMER INDEPENDENT VOWEL QAA are deprecated, and their use is strongly discouraged. One feature of the Khmer script is the introduction of the consonant character for a glottal stop (U+17A2 អ KHMER LETTER QA). This made it unnecessary for each initial vowel sound to have its own independent vowel character, although some independent

vowels exist. Neither U+17A3 nor U+17A4 actually exists in the Khmer script. Other related scripts, including the Devanagari script, have independent vowel characters corresponding to them (*a* and *aa*), but they can be transliterated by *khmer letter qa* and *khmer letter qa + khmer vowel aa,* respectively, without ambiguity because these scripts have no consonant character corresponding to the *khmer qa.*

The use of U+17B4 ᴋʜᴍᴇʀ ᴠᴏᴡᴇʟ ɪɴʜᴇʀᴇɴᴛ ᴀǫ and U+17B5 ᴋʜᴍᴇʀ ᴠᴏᴡᴇʟ ɪɴʜᴇʀᴇɴᴛ ᴀᴀ is discouraged. These newly invented characters do not exist in the Khmer script. They were intended to be used to represent a phonetic difference not expressed by the spelling, so as to assist in phonetic sorting. However, they are insufficient for that purpose and should be considered errors in the encoding. These two characters are ignored by default for collation.

The use of U+17D8 ᴋʜᴍᴇʀ sɪɢɴ ʙᴇʏʏᴀʟ is discouraged. It was supposed to represent "et cetera" in Khmer. However, it is a word rather than a symbol. Moreover, it has several different spellings. It should be spelled out fully using normal letters. *Beyyal* can be written as follows:

᧪ᦖᦖ᧪ *khan + ba + e + khan*

−ᦖᦖ− *en dash + ba + e + en dash*

᧪ᦖ᧪ *khan + lo + khan*

−ᦖ− *en dash + lo + en dash*

Ordering of Syllable Components. The standard order of components in an orthographic syllable as expressed in BNF is

$$\text{B \{R | C\} \{S \{R\}\}* \{\{Z\} V\} \{O\} \{S\}}$$

where

B is a base character (consonant character, independent vowel character, and so on)

R is a *robat*

C is a consonant shifter

S is a subscript consonant or independent vowel sign

V is a dependent vowel sign

Z is a zero width non-joiner or a zero width joiner

O is any other sign

For example, the common word ខ្ញុំ *khnyom* "I" is composed of the following three elements: (1) consonant character *kha* as B; (2) subscript consonant sign *coeng nyo* as S; and (3) dependent vowel sign *om* as V. In the Unicode Standard, *coeng nyo* and *om* are further decomposed, and the whole word is represented by five coded characters.

ខ្ញុំ *kha* + *coeng* + *nyo* + *u* + *nikahit* [kʰɲom] "I"

The order of coded characters does not always match the visual order. For example, some of the dependent vowel signs and their fragments may seem to precede a consonant character, but they are always put after it in the sequence of coded characters. This is also the case with *coeng ro*. Examples of visual reordering and other aspects of syllabic order are shown in *Figure 16-3*.

Figure 16-3. Examples of Syllabic Order in Khmer

តេ *to* + *e* [tèː] "much"

ច្រើន *ca* + *coeng* + *ro* + *oe* + *no* [craən] "much"

សង្គ្រាម *sa* + *ngo* + *coeng* + *ko* + *coeng* + *ro* + *aa* + *mo* [sɔŋkrèəm] "war"

ហើយ *ha* + *oe* + *coeng* + *yo* [haəi] "already"

សញ្ញា *sa* + *nyo* + *coeng* + *nyo* + *aa* [saɲɲaː] "sign"

ស៊ី *sa* + *triisap* + *ii* [siː] "eat"

ប៊ី *ba* + *muusikatoan* + *ii* [pei] "a kind of flute"

Consonant Shifters. U+17C9 ◌៉ KHMER SIGN MUUSIKATOAN and U+17CA ◌៊ KHMER SIGN TRIISAP are consonant shifters, also known as register shifters. In the presence of other superscript glyphs, both of these signs are usually rendered with the same glyph shape as that of U+17BB ◌ុ KHMER VOWEL SIGN U, as shown in the last two examples of *Figure 16-3*.

Although the consonant shifter in handwriting may be written after the subscript, the consonant shifter should always be encoded immediately following the base consonant, except when it is preceded by U+200C ZWNJ ZERO WIDTH NON-JOINER. This provides Khmer with a fixed order of character placement, making it easier to search for words in a document.

ម៉្ងៃ *mo* + *muusikatoan* + *coeng* + *ngo* + *ai* [mŋai] "one day"

ម៊្ហែតៗ *mo* + *triisap* + *coeng* + *ha* + *ae* + *ta* + *lek too* [mhɛːtmhɛːt] "bland"

If either *muusikatoan* or *triisap* needs to keep its superscript shape (as an exception to the general rule that states other superscripts typically force the alternative

subscript glyph for either character), U+200C ᵃᵇ ZERO WIDTH NON-JOINER should be inserted before the consonant shifter to show the normal glyph for a consonant shifter when the general rule requires the alternative glyph. In such cases, U+200C ᵃᵇ ZERO WIDTH NON-JOINER is inserted before the vowel sign, as shown in the following examples:

បៀរ *ba* + ᵃᵇ + *triisap* + *ii* + *yo* + *ae* + *ro* [bijɛ̀ː] "beer"

ប្រតឹងអឺះ *ba* + *coeng* + *ro* + *ta* + *yy* + *ngo* + *qa* + ᵃᵇ + *triisap* + *y* + *reahmuk* [prɔtəːŋʔɯh] "urgent, too busy"

ប្រតឹងអឺះ *ba* + *coeng* + *ro* + *ta* + *yy* + *ngo* + *qa* + *triisap* + *y* + *reahmuk*

Ligature Control. In the *aksaa muul* font style, some vowel signs ligate with the consonant characters to which they are applied. The font tables should determine whether they form a ligature; ligature use in *muul* fonts does not affect the meaning. However, U+200C ᵃᵇ ZERO WIDTH NON-JOINER may be inserted before the vowel sign to explicitly suppress such a ligature, as shown in *Figure 16-4* for the word "savant," pronounced [vituː].

Figure 16-4. Ligation in *Muul* Style in Khmer

វិទ្ធុ	*vo* + *i* + *to* + *uu*	(*aksaa crieng* font)
ទទ្ធុ, វិទ្ធុ	*vo* + *i* + *to* + *uu*	(ligature dependent on the *muul* font)
វិទ្ធុ	*vo* + ᵃᵇ + *i* + *to* + *uu*	(ᵃᵇ to prevent the ligature in a *muul* font)
ទទ្ធុ	*vo* + ᶻʸ + *i* + *to* + *uu*	(ᶻʸ to request the ligature in a *muul* font)

Spacing. Khmer does not use whitespace between words, although it does use whitespace between clauses and between parts of a name. If word boundary indications are desired—for example, as part of automatic line layout algorithms—the character U+200B ˢᵖ ZERO WIDTH SPACE should be used to place invisible marks for such breaks. The ZERO WIDTH SPACE can grow to have a visible width when justified. See *Table 23-2*.

16.4.3 Khmer Symbols: U+19E0–U+19FF

Symbols. Many symbols for punctuation, digits, and numerals for divination lore are encoded as independent entities. Symbols for the lunar calendar are encoded as single characters that cannot be decomposed even if their appearance might seem to be decomposable. U+19E0 ⸗ KHMER SYMBOL PATHAMASAT represents the first *ashadha* (eighth month) of the lunar calendar. During the type of leap year in the lunar calendar known as *adhikameas*, there is also a second *ashadha*. U+19F0 ⸗ KHMER SYMBOL TUTEYASAT represents that second *ashadha*. The 15 characters from U+19E1 ⸗ KHMER SYMBOL MUOY KOET to U+19EF ⸗ KHMER SYMBOL DAP-PRAM

KOET represent the first through the fifteenth lunar waxing days, respectively. The 15 characters from U+19F1 ᧱ KHMER SYMBOL MUOY ROC through U+19FF ᧿ KHMER SYMBOL DAP-PRAM ROC represent the first through the fifteenth waning days, respectively. The typographical form of these lunar dates is a top and bottom section of the same size text. The dividing line between the upper and lower halves of the symbol is the vertical center of the line height.

16.5 Tai Le

16.5.1 Tai Le: U+1950–U+197F

The Tai Le script has a history of 700–800 years, during which time several orthographic conventions were used. The modern form of the script was developed in the years following 1954; it rationalized the older system and added a systematic representation of tones with the use of combining diacritics. The new system was revised again in 1988, when spacing tone marks were introduced to replace the combining diacritics. The Unicode encoding of Tai Le handles both the modern form of the script and its more recent revision.

The Tai Le language is also known as Tai Nüa, Dehong Dai, Tai Mau, Tai Kong, and Chinese Shan. *Tai Le* is a transliteration of the indigenous designation, ᥖᥭᥰ ᥘᥴ [tai^2 lə^6] (in older orthography ᥖᥭᥰ ᥘᥴ). The modern Tai Le orthographies are straightforward: initial consonants precede vowels, vowels precede final consonants, and tone marks, if any, follow the entire syllable. There is a one-to-one correspondence between the tone mark letters now used and existing nonspacing marks in the Unicode Standard. The tone mark is the last character in a syllable string in both orthographies. When one of the combining diacritics follows a tall letter ᥘ, ᥙ, ᥛ, ᥜ, ᥝ or ᥟ, it is displayed to the right of the letter, as shown in *Table 16-11*.

Table 16-11. Tai Le Tone Marks

Syllable	New Orthography	Old Orthography
ta	ᥖ	ᥖ
ta²	ᥖᥰ	ᥖ̈
ta³	ᥖᥱ	ᥖ̌
ta⁴	ᥖᥲ	ᥖ̀
ta⁵	ᥖᥳ	ᥖ̇
ta⁶	ᥖᥴ	ᥖ́
ti	ᥖᥤ	ᥖᥤ
ti²	ᥖᥤᥰ	ᥖᥤ̈
ti³	ᥖᥤᥱ	ᥖᥤ̌
ti⁴	ᥖᥤᥲ	ᥖᥤ̀
ti⁵	ᥖᥤᥳ	ᥖᥤ̇
ti⁶	ᥖᥤᥴ	ᥖᥤ́

Digits. In China, European digits (U+0030..U+0039) are mainly used, although Myanmar digits (U+1040..U+1049) are also used with slight glyph variants. Note the differences, in particular, for the digits 2, 6, 8, and 9, as shown in *Table 16-12*.

Table 16-12. Myanmar Digits in Tai Le

Value	Myanmar	Tai Le
0	၀	၀
1	၁	၁
2	၂	၂
3	၃	၃
4	၄	၄
5	၅	၅
6	၆	၆
7	၇	၇
8	၈	၈
9	၉	၉

Punctuation. Both CJK punctuation and Western punctuation are used. Typographically, European digits are about the same height and depth as the tall characters ၊ and ၊. In some fonts, the baseline for punctuation is the depth of those characters.

16.6 New Tai Lue

16.6.1 New Tai Lue: U+1980–U+19DF

The New Tai Lue script, also known as Xishuangbanna Dai, is used mainly in southern China. The script was developed in the 20th century as an orthographic simplification of the historic Lanna script used to write the Tai Lue language. "Lanna" refers to a region in present-day northern Thailand as well as to a Tai principality that existed in that region from approximately the late thirteenth century to the early 20th century. The Lanna script grew out of the Mon script and was adapted in various forms in the Lanna kingdom and by Tai-speaking communities in surrounding areas that had close contact with the kingdom, including southern China. The Lanna script, also known as the Tai Tham script (see *Section 16.7, Tai Tham*), is still used to write various languages of the Tai family today, including Tai Lue. The approved orthography for this language uses the New Tai Lue script; however, usage of the older orthography based on a variant of Lanna script can still be found.

New Tai Lue differs from Tai Tham in that it regularizes the consonant repertoire, simplifies the writing of consonant clusters and syllable-final consonants, and uses only spacing vowel signs, which appear before or after the consonants they modify, and which are stored in visual order. By contrast, Lanna uses both spacing vowel signs and nonspacing vowel signs, which appear above or below the consonants they modify, and all of which are stored in logical order.

Structure. New Tai Lue is written left to right. Despite its simplification from the Tai Tham (Lanna) script, it retains an important feature of *abugidas*: the consonant letters have the inherent vowel /a/, which is modified to some other vowel by the addition of an explicit vowel letter.

Visual Order. The New Tai Lue script uses visual ordering—a characteristic it shares with the Thai and Lao scripts. This means that the four New Tai Lue vowels that occur visually on the left side of their associated consonant are stored ahead of those consonants in text. This practice differs from the usual pattern for Brahmi-derived scripts, in which all dependent vowels are stored in logical order after their associated consonants, even when they are displayed to the left of those consonants.

Visual order for New Tai Lue vowels results in simpler rendering for the script and follows current accepted practice for data entry. However, it complicates syllable identification and the processes for searching and sorting. Implementers can take advantage of techniques developed for processing Thai script data to address the issues associated with visual order encoding.

The four New Tai Lue vowel letters that occur in visual order ahead of their associated consonants are given the property value Logical_Order_Exception = True in the Unicode Character Database.

Implementers should note that the visual order model for New Tai Lue was formally introduced as of Unicode 8.0. When New Tai Lue was added to the Unicode Standard in Version 4.1, the text model for the script followed the normal Indic practice: all dependent vowels were intended to follow their consonant, regardless of visual placement. However, in practice, the majority of New Tai Lue text data using Unicode characters prior to Unicode 8.0 already uses visual ordering, and many extant New Tai Lue fonts also assume visual ordering. As a result, the model change for New Tai Lue as of Unicode 8.0 should not pose a substantial migration issue for data or fonts. However, implementations may have glitches in some algorithmic behavior until underlying libraries and platform support catch up to the character property changes for New Tai Lue as of Unicode 8.0 or later versions.

Two-Part Vowels. Some vowels in New Tai Lue are represented with two vowel letters—one to the left of the consonant letter and one to the right. In these cases, the characters are simply stored in visual order: first the vowel letter on the left, then the consonant letter, and finally the vowel letter on the right. U+19B6 NEW TAI LUE VOWEL SIGN AE is considered a single letter and is displayed to the left of its consonant letter. It is not represented by a sequence of two characters for U+19B5 NEW TAI LUE VOWEL SIGN E. If a tone mark appears in a syllable, it occurs last in the representation, after any right side vowel, again in visual order. *Table 16-13* shows several examples of these ordering relations.

Table 16-13. New Tai Lue Vowel Placement

ᩅ	e	+	ᩈ	ka	+	ᦖ	t1				→	ᩅᩈᦖ	[ke:²]
ᩅ	e	+	ᩈ	ka	+	θ	i				→	ᩅᩈθ	[kə:¹]
ᩅ	e	+	ᩈ	ka	+	θᒿ	iy				→	ᩅᩈθᒿ	[kəi¹]
ᩅ	e	+	ᩈ	ka	+	θᒿ	iy	+	ᦖ	t1	→	ᩅᩈθᒿᦖ	[kəi²]
ᩅ	e	+	ᩈ	ka	+	θᒿ	iy	+	e	t2	→	ᩅᩈθᒿe	[kəi³]

Final Consonants. A virama or killer character is not used to create conjunct consonants in New Tai Lue, because clusters of consonants do not regularly occur. New Tai Lue has a limited set of final consonants, which are modified with a hook showing that the inherent vowel is killed.

Tones. Similar to the Thai and Lao scripts, New Tai Lue consonant letters come in pairs that denote two tonal registers. The tone of a syllable is indicated by the combination of the tonal register of the consonant letter plus a tone mark written at the end of the syllable, as shown in *Table 16-14*.

Table 16-14. New Tai Lue Registers and Tones

Display	Sequence	Register	Tone Mark	Tone	Transcription
ꪈ	ka^h	high		1	[ka^1]
ꪈꪱ	ka^h + t1	high	t1	2	[ka^2]
ꪈꫀ	ka^h + t2	high	t2	3	[ka^3]
ꪁ	ka^l	low		4	[ka^4]
ꪁꪱ	ka^l + t1	low	t1	5	[ka^5]
ꪁꫀ	ka^l + t2	low	t2	6	[ka^6]

Digits. The New Tai Lue script adapted its digits from the Tai Tham (or Lanna) script. Tai Tham used two separate sets of digits, one known as the *hora* set, and one known as the *tham* set. The New Tai Lue digits are adapted from the *hora* set.

The one exception is the additional New Tai Lue digit for one: U+19DA ꧚ NEW TAI LUE THAM DIGIT ONE. The regular *hora* form for the digit, U+19D1 ꧑ NEW TAI LUE DIGIT ONE, has the exact same glyph shape as a common New Tai Lue vowel, U+19B1 ꪱ NEW TAI LUE VOWEL SIGN AA. For this reason, U+19DA is often substituted for U+19D1 in contexts which are not obviously numeric, to avoid visual ambiguity. Implementations of New Tai Lue digits need to be aware of this usage, as U+19DA may occur frequently in text.

16.7 Tai Tham

16.7.1 Tai Tham: U+1A20–U+1AAF

The script called Tai Tham is used for three living languages, Lue, Khuen, and Northern Thai, which are spoken in China, Myanmar, Northern Thailand, and surrounding areas. In addition, the script is used for Lao Tham (or Old Lao) and other dialect variants found in Buddhist palm leaves and notebooks. Although the script has no single, commonly recognized name across the region today, it is known by various language-specific and region-specific names, such as Old Xishuangbanna Dai or Old Tai Lue in China, Khün in Myanmar, and Tua Mueang, Lanna, or Yuan in Thailand.

Few of the six million speakers of Northern Thai are literate in the Tai Tham script, although there is some rising interest in the script among the young. There are about 690,000 speakers of Tai Lue. Of those, many people born before 1950 are literate in the Tai Tham script, and newspapers and other literature are regularly produced in the Xishuangbanna region of Yunnan using the script. Younger speakers are taught the New Tai Lue script, instead. (See *Section 16.6, New Tai Lue.*) The Tai Tham script continues to be taught in the Tai Lue monasteries. There are 107,000 speakers of Khün, for which Tai Tham is the only script.

Consonants. Consonants have an inherent -*a* vowel sound. Most consonants have a combining subjoined form, but unlike most other Brahmi-derived scripts, the subjoining of a consonant does not mean that the vowel of the previous consonant is killed. A subjoined consonant may be the first consonant of the following syllable. The encoding model for Tai Tham is more similar to the Khmer *coeng* model than to the usual virama model: the character U+1A60 TAI THAM SIGN SAKOT is entered before a consonant which is to take the subjoined form. A subjoined consonant may be attached to a dependent vowel sign.

U+1A4B TAI THAM LETTER A represents a glottal consonant. Its rendering in Northern Thai differs from that typical for Tai Lue and Khün.

A number of Tai Tham characters did not traditionally take subjoined forms, but modern innovations in borrowed vocabulary suggest that fonts should make provision for subjoining behavior for all of the consonants except the historical *vocalic r* and *l*.

Independent Vowels. Independent vowels are used as in other Brahmi-derived scripts. U+1A52 TAI THAM LETTER OO is not used in Northern Thai.

Dependent Consonant Signs. Seven dependent consonant signs occur. Two of these are used as medials: U+1A55 TAI THAM CONSONANT SIGN MEDIAL RA and U+1A56

TAI THAM CONSONANT SIGN MEDIAL LA form clusters and immediately follow a consonant.

U+1A58 TAI THAM SIGN MAI KANG LAI is used as a final *-ng* in Northern Thai and Tai Lue. Its shape is distinct in Khün. U+1A59 TAI THAM CONSONANT SIGN FINAL NGA is also used as a final *-ng* in Northern Thai.

U+1A5B TAI THAM CONSONANT SIGN HIGH RATHA OR LOW PA represents *high ratha* in *santhān* "shape" and *low pa* in *sappa* "omniscience".

Dependent Vowel Signs. Dependent vowel signs are used in a manner similar to that employed by other Brahmi-derived scripts, although Tai Tham uses many of them in combination.

U+1A63 TAI THAM VOWEL SIGN AA and U+1A64 TAI THAM VOWEL SIGN TALL AA are separately encoded because the choice of which form to use cannot be reliably predicted from context.

The Khün character U+1A6D TAI THAM VOWEL SIGN OY is not used in Northern Thai. Khün vowel order is quite different from that of Northern Thai.

Tone Marks. Tai Tham has two combining tone marks, U+1A75 TAI THAM SIGN TONE-1 and U+1A76 TAI THAM SIGN TONE-2, which are used in Tai Lue and in Northern Thai. These are rendered above the vowel over the base consonant. Three additional tone marks are used in Khün: U+1A77 TAI THAM SIGN KHUEN TONE-3, U+1A78 TAI THAM SIGN KHUEN TONE-4, and U+1A79 TAI THAM SIGN KHUEN TONE-5, which are rendered above and to the right of the vowel over the base consonant. Tone marks are represented in logical order following the vowel over the base consonant or consonant stack. If there is no vowel over a base consonant, then the tone is rendered directly over the consonant; this is the same way tones are treated in the Thai script.

Other Combining Marks. U+1A7A TAI THAM SIGN RA HAAM is used in Northern Thai to indicate that the character or characters it follows are not sounded. The precise range of characters not to be sounded is indeterminant; it is defined instead by reading rules. In Tai Lue, *ra haam* is used as a final *-n*.

The mark U+1A7B TAI THAM SIGN MAI SAM has a range of uses in Northern Thai:

- It is used as a repetition mark, stored as the last character in the word to be repeated: *tang* "be different", *tangtang* "be different in my view".

- It is used to disambiguate the use of a subjoined letters. A subjoined letter may be a medial or final, or it may be the start of a new syllable.

- It is used to mark "double-acting" consonants. It is stored where the consonant would be stored if there were a separate consonant used.

U+1A7F TAI THAM COMBINING CRYPTOGRAMMIC DOT is used singly or multiply beneath letters to give each letter a different value according to some hidden agreement between reader and writer.

Digits. Two sets of digits are in common use: a secular set (Hora) and an ecclesiastical set (Tham). European digits are also found in books.

Punctuation. The four signs U+1AA8 TAI THAM SIGN KAAN, U+1AA9 TAI THAM SIGN KAANKUU, U+1AAA TAI THAM SIGN SATKAAN, and U+1AAB TAI THAM SIGN SATKAANKUU, are used in a variety of ways, with progressive values of finality. U+1AAB TAI THAM SIGN SATKAANKUU is similar to U+0E5A THAI CHARACTER ANGKHANKHU.

At the end of a section, U+1AA9 TAI THAM SIGN KAANKUU and U+1AAC TAI THAM SIGN HANG may be combined with U+1AA6 TAI THAM SIGN REVERSED ROTATED RANA in a number of ways. The symbols U+1AA1 TAI THAM SIGN WIANGWAAK, U+1AA0 TAI THAM SIGN WIANG, and U+1AA2 TAI THAM SIGN SAWAN are logographs for "village," "city," and "heaven," respectively.

The three signs U+1AA3 TAI THAM SIGN KEOW, "courtyard," U+1AA4 TAI THAM SIGN HOY, "oyster," and U+1AA5 TAI THAM SIGN DOKMAI, "flower" are used as dingbats and as section starters. The mark U+1AA7 TAI THAM SIGN MAI YAMOK is used in the same way as its Thai counterpart, U+0E46 THAI CHARACTER MAIYAMOK.

European punctuation like question mark, exclamation mark, parentheses, and quotation marks is also used.

Collating Order. There is no firmly established sorting order for the Tai Tham script. The order in the code charts is based on Northern Thai and Thai. U+1A60 TAI THAM SIGN SAKOT is ignored for sorting purposes.

Line Breaking. Opportunities for line breaking are lexical, but a line break may not be inserted between a base letter and a combining diacritic. There is no insertion of visible hyphens at line boundaries.

16.8 Tai Viet

16.8.1 Tai Viet: U+AA80–U+AADF

The Tai Viet script is used by three Tai languages spoken primarily in northwestern Vietnam, northern Laos, and central Thailand: Tai Dam (also Black Tai or Tai Noir), Tai Dón (White Tai or Tai Blanc), and Thai Song (Lao Song or Lao Song Dam). The Thai Song of Thailand are geographically removed from, but linguistically related to the Tai people of Vietnam and Laos. There are also populations in Australia, China, France, and the United States. The script is related to other Tai scripts used throughout Southeast Asia. The total population using the three languages, across all countries, is estimated to be 1.3 million (Tai Dam 764,000, Tai Dón 490,000, Thai Song 32,000). The script is still used by the Tai people in Vietnam, and there is a desire to introduce it into formal education there. It is unknown whether it is in current use in Laos, Thailand, or China.

Several different spellings have been employed for the name of the script, including Tay Viet. Linguists commonly use "Thai" to indicate the language of central Thailand, and "Tai" to indicate the language family; however, even that usage is inconsistent.

Structure. The Tai Viet script shares many features with other Tai alphabets. It is written left to right and has a double set of initial consonants, one for the low tone class and one for the high tone class. Vowels marks are positioned before, after, above, or below the syllable's initial consonant, depending on the vowel. Some vowels are written with digraphs. The consonants do not carry an implicit vowel. The vowel must always be written explicitly.

The Tai languages are almost exclusively monosyllabic. A very small number of words have an unstressed initial syllable, and loan words may be polysyllabic.

Visual Order. The Tai Viet script uses visual ordering—a characteristic it shares with the Thai and Lao scripts. This means that the five Tai Viet vowels that occur visually on the left side of their associated consonant are stored ahead of those consonants in text. This practice differs from the usual pattern for Brahmi-derived scripts, in which all dependent vowels are stored in logical order after their associated consonants, even when they are displayed to the left of those consonants.

Visual order for Tai Viet vowels results in simpler rendering for the script and follows accepted practice for data entry. However, it complicates syllable identification and the processes for searching and sorting. Implementers can take advantage of techniques developed for processing Thai script data to address the issues associated with visual order encoding.

The five Tai Viet vowels that occur in visual order ahead of their associated consonants are given the property value Logical_Order_Exception = True in the Unicode Character Database.

Tone Classes and Tone Marks. In the Tai Viet script each consonant has two forms. The low form of the initial consonant indicates that the syllable uses tone 1, 2, or 3. The high form of the initial consonant indicates that the syllable uses tone 4, 5, or 6. This is sufficient to define the tone of closed syllables (those ending /p/, /t/, /k/, or /ʔ/), in that these syllables are restricted to tones 2 and 5.

Traditionally, the Tai Viet script did not use any further marking for tone. The reader had to determine the tone of unchecked syllables from the context. Recently, several groups have introduced tone marks into Tai Viet writing. Tai Dam speakers in the United States began using Lao tone marks with their script in the 1970s, and those marks are included in SIL's Tai Heritage font. These symbols are written as combining marks above the initial consonant, or above a combining vowel, and are identified by their Laotian names, *mai ek* and *mai tho*. These marks are also used by the Song Petburi font (developed for the Thai Song language), although they were probably borrowed from the Thai alphabet rather than the Lao.

The Tai community in Vietnam invented their own tone marks written on the base line at the end of the syllable, which they call *mai nueng* and *mai song*.

When combined with the consonant class, two tone marks are sufficient to unambiguously mark the tone. No tone is written on loan words or on the unstressed initial syllable of a native word.

Final Consonants. U+AA9A TAI VIET LETTER LOW BO and U+AA92 TAI VIET LETTER LOW DO are used to write syllable-final /p/ and /t/, respectively, as is the practice in many Tai scripts. U+AA80 TAI VIET LETTER LOW KO is used for both final /k/ and final /ʔ/. The high-tone class symbols are used for writing final /j/ and the final nasals, /m/, /n/, and /ŋ/. U+AAAB TAI VIET LETTER HIGH VO is used for final /w/.

There are a number of exceptions to the above rules in the form of vowels which carry an inherent final consonant. These vary from region to region. The ones included in the Tai Viet block are the ones with the broadest usage: /-aj/, /-am/, /-an/, and /-əw/.

Symbols and Punctuation. There are five special symbols in Tai Viet. The meaning and use of these symbols is summarized in *Table 16-15*.

Table 16-15. Tai Viet Symbols and Punctuation

Code	Glyph	Name	Meaning
AADB	ꦕꦵ	*kon*	person

AADC	ꫜ	*nueng*	one
AADD	ꫝ	*sam*	signals repetition of the previous word
AADE	꫞	*ho hoi*	beginning of text (used in songs and poems)
AADF	꫟	*koi koi*	end of text (used in songs and poems)

U+AADB TAI VIET SYMBOL KON and U+AADC TAI VIET SYMBOL NUENG may be regarded as word ligatures. They are, however, encoded as atomic symbols, without decompositions. In the case of *kon*, the word ligature symbol is used to distinguish the common word "person" from otherwise homophonous words.

Word Spacing. Traditionally, the Tai Viet script was written without spaces between words. In the last thirty years, users in both Vietnam and the United States have started writing spaces between words, in both handwritten and machine produced texts. Most users now use interword spacing. Polysyllabic words may be written without space between the syllables.

Collating Order. The Tai Viet script does not have an established standard for sorting. Sequences have sometimes been borrowed from neighboring languages. Some sources use the Lao order, adjusted for differences between the Tai Dam and Lao character repertoires. Other sources prefer an order based on the Vietnamese alphabet. It is possible that communities in different countries will want to use different orders.

16.9 Kayah Li

16.9.1 Kayah Li: U+A900–U+A92F

The Kayah Li script was invented in 1962 by Htae Bu Phae (also written Hteh Bu Phe), and is used to write the Eastern and Western Kayah Li languages of Myanmar and Thailand. The Kayah Li languages are members of the Karenic branch of the Sino-Tibetan family, and are tonal and mostly monosyllabic. There is no mutual intelligibility with other Karenic languages.

The term *Kayah Li* is an ethnonym referring to a particular Karen people who speak these languages. *Kayah* means "person" and *li* means "red," so *Kayah Li* literally means "red Karen." This use of color terms in ethnonyms and names for languages is a common pattern in this part of Southeast Asia.

Structure. Although Kayah Li is a relatively recently invented script, its structure was clearly influenced by Brahmi-derived scripts, and in particular the Myanmar script, which is used to write other Karenic languages. The order of letters is a variant of the general Brahmic pattern, and the shapes and names of some letters are Brahmi-derived. Other letters are innovations or relate more specifically to Myanmar-based orthographies.

The Kayah Li script resembles an *abugida* such as the Myanmar script, in terms of the derivation of some vowel forms, but otherwise Kayah Li is closer to a true alphabet. Its consonants have no inherent vowel, and thus no virama is needed to remove an inherent vowel.

Vowels. Four of the Kayah Li vowels (a, ơ, i, ô) are written as independent spacing letters. Five others (ư, e, u, ê, o) are written by means of diacritics applied above the base letter U+A922 KAYAH LI LETTER A, which thus serves as a vowel-carrier. The same vowel diacritics are also written above the base letter U+A923 KAYAH LI LETTER OE to represent sounds found in loanwords.

Tones. Tone marks are indicated by combining marks which subjoin to the four independent vowel letters. The vowel diacritic U+A92A KAYAH LI VOWEL O and the mid-tone mark, U+A92D KAYAH LI TONE CALYA PLOPHU, are each analyzable as composite signs, but encoding of each as a single character in the standard reflects usage in didactic materials produced by the Kayah Li user community.

Digits. The Kayah Li script has its own set of distinctive digits.

Punctuation. Kayah Li text makes use of modern Western punctuation conventions, but the script also has two unique punctuation marks: U+A92E KAYAH LI SIGN CWI and U+A92F KAYAH LI SIGN SHYA. The *shya* is a script-specific form of a *danda* mark.

16.10 Cham

16.10.1 Cham: U+AA00–U+AA5F

Cham is a Austronesian language of the Malayo-Polynesian family. The Cham language has two major dialects: Eastern Cham and Western Cham. Eastern Cham speakers live primarily in the southern part of Vietnam and number about 73,000. Western Cham is spoken mostly in Cambodia, with about 220,000 speakers there and about 25,000 in Vietnam. The Cham script is used more by the Eastern Cham community.

Structure. Cham is a Brahmi-derived script. Consonants have an inherent vowel. The inherent vowel is -*a* in the case of most consonants, but is -*u* in the case of nasal consonants. There is no virama and hence no killing of the inherent vowel. Dependent vowels (matras) are used to modify the inherent vowel and separately encoded, explicit final consonants are used where there is no inherent vowel. The script does not have productive formation of consonant conjuncts.

Independent Vowel Letters. Six of the initial vowels in Cham are represented with unique, independent vowels. These separately-encoded characters always indicate a syllable-initial vowel, but they may occur word-internally at a syllable break. Other Cham vowels which do not have independent forms are instead represented by dependent vowels (matras) applied to U+AA00 CHAM LETTER A. Four of the other independent vowel letters are also attested bearing matras.

Consonants. Cham consonants can be followed by consonant signs to represent the glides: -*ya*, -*ra*, -*la*, or -*wa*. U+AA33 CHAM CONSONANT SIGN YA, in particular, normally ligates with the base consonant it modifies. When it does so, any dependent vowel is graphically applied to it, rather than to the base consonant.

The independent vowel U+AA00 CHAM LETTER A can cooccur with two of the medial consonant signs: -*ya* or -*wa*. The writing system distinguishes these sequences from single letters which are pronounced the same. Thus, <a, -ya> [ja] contrasts with U+AA22 CHAM LETTER YA, also pronounced [ja], and <a, -wa> [wa] contrasts with U+AA25 CHAM LETTER VA, also pronounced [wa].

Four medial clusters of two consonant signs in a row occur: <-ra, -ya> [-rja], <-ra, -wa> [-rwa], <-la, -ya> [-lja], and <-la, -wa> [-lwa].

There are three types of final consonants. The majority are simply encoded as separate base characters. Graphically, those final forms appear similar to the corresponding non-final consonants, but typically have a lengthened stroke at the right side of their glyphs. The second type consist of combining marks to represent final -*ng*, -*m*, and -*h*. Finally, U+AA25 CHAM LETTER VA occurs unchanged either in initial

or final positions. Final consonants may occur word-internally, in which case they indicate the presence of a syllable boundary.

Ordering of Syllable Components. Dependent vowels and other signs are encoded after the consonant to which they apply. The ordering of elements is shown in more detail in *Table 16-16*.

<p align="center">Table 16-16. Cham Syllabic Structure</p>

Class	Examples	Encoding
consonant or independent vowel	ꨀ	[U+AA00..U+AA28]
consonant sign -ra, -la	ꨴꯨ,ꨵ	[U+AA34, U+AA35]
consonant sign -ya, -wa	ꨳ,ꨶ	[U+AA33, U+AA36]
left-side dependent vowel	ꨯ,ꨰ	[U+AA2F, U+AA30]
other dependent vowel	ꨩ	[U+AA2A..U+AA2E, U+AA31..U+AA32]
vowel lengthener -aa	ꨩ	U+AA29
final consonant or va	ꨮ,ꩍ	[U+AA40..U+AA4D, U+AA25]

The left-side dependent vowels U+AA2F CHAM VOWEL SIGN O and U+AA30 CHAM VOWEL SIGN AI occur in logical order after the consonant (and any medial consonant signs), but in visual presentation their glyphs appear *before* (to the left of) the consonant. U+AA2F CHAM VOWEL SIGN O, in particular, may occur together in a sequence with another dependent vowel, the vowel lengthener, or both. In such cases, the glyph for U+AA2F appears to the left of the consonant, but the glyphs for the second dependent vowel and the vowel lengthener are rendered above or to the right of the consonant.

Digits. The Cham script has its own set of digits, which are encoded in this block. However, European digits are also known and occur in Cham texts because of the influence of Vietnamese.

Punctuation. Cham uses *danda* marks to indicate text units. Three levels are recognized, marked respectively with *danda*, *double danda*, and *triple danda*.

U+AA5C CHAM PUNCTUATION SPIRAL often begins a section of text. It can be compared to the usage of Tibetan head marks. The *spiral* may also occur in combination with a *danda*.

Modern Cham text also makes use of European punctuation marks, such as the question mark, hyphen and colon.

Line Breaking. Opportunities for line breaks occur after any full orthographic syllable in Cham. Modern Cham text makes use of spaces between words, and those are also line break opportunities. Line breaks occur after *dandas*.

16.11 Pahawh Hmong

16.11.1 Pahawh Hmong: U+16B00–U+16B8F

The Pahawh Hmong script was originally devised by Shong Lue Yang in 1959 to write the Hmong language. The script was devised in Laos, and taken to refugee camps in northern Thailand. In the late 20th century, it then moved with waves of immigrants to Australia and the United States, where it remains in current use. The Hmong language is also commonly written using the Romanized Popular Alphabet (RPA), which uses the Latin script.

The Pahawh Hmong writing system has had four stages of development: the Source version, the Second Stage Reduced Version, the Third Stage Reduced Version, and the Final Version. Only the Second and Third Stage versions are in current use. The characters in the Pahawh Hmong block support text written in the Second Stage Reduced, Third Stage Reduced, and Final versions.

Character Names. The Pahawh Hmong character names are based on the Third Stage Reduced Version, which provides a one-to-one mapping between the Third Stage tone diacritics and the widely used Romanized Popular Alphabet tone mark letters. The Second Stage names are listed as annotations in the names list.

Structure. The Pahawh Hmong script is written left to right. Text consists of a sequence of syllables that may contain a maximum length of four characters (two base characters and two diacritics). Syllables are separated by spaces.

Unlike other writing systems, Pahawh Hmong writes the vowel of a syllable before the syllable-initial consonant, as illustrated in *Figure 16-5*.

Figure 16-5. Pahawh Hmong Syllable Structure

ə + ◌̇ + K + ◌̌ → ə̇Ǩ

16B16   16B30   16B1D   16B35

The example in *Figure 16-5* uses Second Stage Reduced Version conventions. The representation of the syllable is in straightforward visual order. U+16B16 PAHAWH HMONG VOWEL KAB is the base character representing the [a] vowel of the syllable. The combining mark U+16B30 represents the tone mark for the vowel. U+16B1D PAHAWH HMONG CONSONANT NTSAU is the base character representing the initial consonant of the syllable. The combining mark U+16B35 is a diacritical mark which changes the sound of the consonant from [^{n}ts] to [p^h]. Altogether, the sequence represents the syllable [p^{h}â].

Because the order of characters in memory matches the visual written order of the text, display rendering does not require any reordering of glyphs. However, implementations such as text-to-speech need to be aware that Pahawh Hmong has unusual reading rules, because initial consonants for syllables graphically follow the vowels which they precede in pronunciation.

Vowels. The characters in the range U+16B00..U+16B1B represent vowels. The addition of a diacritic alters the tone of the vowel. The special characters U+16B1A PAHAWH HMONG VOWEL KAAB and U+16B1B PAHAWH HMONG VOWEL KAAV are atomic characters and do not decompose.

Consonants. U+16B1C..U+16B2F represent consonants. These are phonologically initial in a syllable, but occur after the vowel in written order.

Combining Marks. The combining marks in the range U+16B30..U+16B36 are used as tone marks. They combine with the vowel letters to indicate particular tones for the syllable. The use for representation of particular tones differs for the two different stages.

U+16B30 PAHAWH HMONG MARK CIM TUB and U+16B35 PAHAWH HMONG MARK CIM HOM also combine with initial consonant letters. When used this way, these marks function as diacritics and indicate a different sound for the consonant letter. Usually the resultant sound is unrelated to that of the unmodified base letter—the particular modification by the diacritic is not predictable.

Punctuation and Other Symbols. Pahawh Hmong makes use of common European punctuation marks as well as script-specific punctuation marks (U+16B37..U+16B3B and U+16B44..U+16B45). The script employs several mathematical operators (U+16B3C..U+16B3F) in simple arithmetic expressions. Those operators are considered script-specific to Pahawh Hmong, and are not part of the repertoire of symbols used in international mathematical notation; hence they are given a General_Category value of Other_Symbol, rather than Math_Symbol.

Pahawh Hmong also includes a set of modifiers that have various uses: U+16B42..U+16B43 indicate reduplication, U+16B40 identifies the chanting nature of a text, and U+16B41 indicates the following syllable has a non-Hmong pronunciation.

Digits and Numbers. The decimal digits 0–9 are encoded from U+16B50..U+16B59. The representative glyph for U+16B50 PAHAWH HMONG DIGIT ZERO resembles an "I", and is found in the Second Stage Reduced Version orthography. In contrast, the Third Stage Reduced Version orthography has a circular glyph.

A non-decimal system also exists in Pahawh Hmong and is taught today, however, it is not used for arithmetic calculation. The non-decimal numbers are encoded in the range from U+16B5B..U+16B61. The Second Stage Reduced Version glyph for

U+16B5B PAHAWH HMONG NUMBER TENS resembles a "W". The Third Stage Reduced Version glyph looks like an "I", and should be distinguished in fonts from U+16B50 PAHAWH HMONG DIGIT ZERO.

Logographs. Characters encoded from U+16B63..U+16B8F are logographs. These include a grammatical classifier (U+16B63). Also included are characters designating periods of time (U+16B64..U+16B6C), correspondence (U+16B6D..U+16B77), and clan names (U+16B7E..U+16B8F). The clan names are encoded for historical reasons, and are not in widespread current use.

16.12 Nyiakeng Puachue Hmong

16.12.1 Nyiakeng Puachue Hmong: U+1E100–U+1E14F

Nyiakeng Puachue Hmong is a writing system created in the 1980s by Reverend Chervang Kong to write the White Hmong and Green Hmong languages. It is also called the Ntawv Txawjvaag or Chervang script, and was devised for use in the United Christians Liberty Evangelical church in the United States. It is reportedly used today in Laos, Thailand, Vietnam, France and Australia. Several of the letters appear to derive from shapes of Hebrew letters.

Structure. Nyiakeng Puachue Hmong is written from left to right. The script consists of an alphabet with letters for both consonants and vowels. It has combining marks to indicate tones. Syllables are often run together, but when a syllable begins with a vowel, a space must precede it.

Vowels. U+1E124..U+1E12C represent vowels. Diphthongs are written as sequences of two vowels.

Consonants. The characters in the range U+1E100..U+1E123 represent consonants. Preaspirated and devoiced consonants are written with U+1E104 NYIAKENG PUACHUE HMONG LETTER HA following the base consonant. Two consonants can be read as vowels: U+1E123 NYIAKENG PUACHUE HMONG LETTER RRA can be read as the diphthong *ai* and U+1E11F NYIAKENG PUACHUE HMONG LETTER HAH can be read as the vowel *o*.

Combining Marks. The combining marks in the range U+1E130..U+1E136 indicate tone. They appear above the central character of a word, whether a consonant or a vowel. If a word has three or more consonants before a vowel, the mark goes on the center of the whole word, usually on the first vowel, although the user has some discretion regarding its exact placement.

Punctuation. Nyiakeng Puachue Hmong text makes use of standard Western punctuation marks.

Determinatives. Characters from U+1E137..U+1E13B are determinatives, which indicate that the preceding noun is the name of a person, place, thing, vertebrate or invertebrate animal, or a pet name for the animal. Determinatives are not pronounced, but help distinguish homophones. They appear as the last character in a word, and are not separated by a space.

Digits. Script-specific digits are used for Nyiakeng Puachue Hmong. They are encoded in the range from U+1E140...U+1E149.

Other Symbols. The character U+1E13C NYIAKENG PUACHUE HMONG SIGN XW XW indicates that the preceding short word or syllable should be repeated. When separated by a space, it can be used to repeat a whole phrase or sentence. Its use is similar to U+0E46 THAI CHARACTER MAIYAMOK and U+0EC6 LAO KO LA.

U+1E13D NYIAKENG PUACHUE HMONG SYLLABLE LENGTHENER indicates that a vowel is lengthened in a word or phrase.

The logogram U+1E14E NYIAKENG PUACHUE HMONG LOGOGRAM NYAJ represents the word for "money, currency," and can be used in place of the word "money" or before a number or amount. The symbol U+1E14F NYIAKENG PUACHUE HMONG CIRCLED CA indicates ownership.

16.13 Pau Cin Hau

16.13.1 Pau Cin Hau: U+11AC0–U+11AFF

The Pau Cin Hau alphabet is a liturgical script of the Laipian religious tradition, which emerged in the Chin Hills region of present-day Chin State, Myanmar at the turn of the 20th century. The script is named after Pau Cin Hau (1859–1948), a Tedim Chin, who founded the Laipian tradition and developed the script to convey his teachings. In an account given by J. J. Bennison in the 1931 Census of India report for Burma, Pau Cin Hau stated that the characters of his script were revealed to him in a dream in 1902.

The script was designed to represent Tedim, a language of the northern branch of the Kuki-Chin group of the Tibeto-Burman family, which is spoken in Chin State. Tedim is the modern name for the language previously known as Tiddim; it also refers to the Tedim dialects Kamhau (Kamhow) and Sokte.

While the script was developed for writing Tedim, several letters and tone marks represent sounds that are not attested in Tedim, but which exist in other Chin languages, suggesting that the alphabet may have been created as a universal script for the Chin languages.

There are two distinct writing systems associated with Pau Cin Hau and the Laipian tradition. One is an obsolete syllabary and the other is the alphabetic system encoded in this block. Both are attested in manuscript and printed sources. The alphabetic script is derived from the syllabary. Neither of these scripts has any genetic relationship with any other writing system.

Structure. The Pau Cin Hau alphabet has 57 characters consisting of 21 consonant letters, 7 vowel letters, 9 final-consonant letters, and 20 tone marks. It is written from left to right. Vowels, consonants, and tone marks are written linearly as independent characters. The syllable canon for Tedim may be described as (C1)V1(V2)(C2)T. The tone (T) is represented using one of the 20 tone marks. These marks are used for indicating vowel length, tone, and glottal stop, as well as punctuation. Of these, 15 represent tones and 5 represent glottal stop. Ten of the tone marks have a dual role and simultaneously denote tone (or glottal stop) and sentence ending.

Digits. Pau Cin Hau uses European digits.

Punctuation. Word boundaries are indicated using spaces. The end of a sentence is marked with final forms of tone marks. Western punctuation is also used. In some cases, sentence-final tone marks may be redundantly followed by a full stop or other Western punctuation mark.

Line breaking should occur at spaces. Words are not broken at end-of-line and no hyphen is used or attested. No breaking may occur between a tone mark and the character that precedes it.

16.14 Hanifi Rohingya

16.14.1 Hanifi Rohingya: U+10D00–U+10D3F

Hanifi Rohingya is a script used to write the Rohingya language, an Indo-Aryan language spoken by approximately one million people primarily in Myanmar and Bangladesh, but also spoken in other countries along the Indian Ocean. The script was developed by the Rohingya Language Committee in the 1980s under the guidance of Maulana Mohammed Hanif. Rohingya is also written using the Myanmar, Arabic, and Latin scripts. The Hanifi Rohingya script is a modern construction that exhibits Arabic influence in the general appearance and structure of the script. Some letter shapes also show influence from Latin and Myanmar. The script is used for the publication of books and newspapers, both handwritten and printed.

Structure. Hanifi Rohingya is an alphabetic script written from right to left. The script is structurally conjoining and is modeled after Arabic. Adjacent letters join at the baseline. Although letters may not be completely connected at the baseline in handwritten texts, connections between letters are consistently maintained in modern printed texts.

Consonant Letters. Consonant letters represent pure consonants. However, the letter-like sign U+10D22 HANIFI ROHINGYA MARK SAKIN is often used optionally to indicate a consonant shown in isolation or a consonant at the end of a word. The consonant gemination sign U+10D27 HANIFI ROHINGYA SIGN TASSI indicates doubled consonants. When both the *tassi* and a tonal sign are present, the *tassi* is placed first and the tonal sign is displayed visually to its left.

Vowels. Vowels are expressed using five letter-like vowel signs (U+10D1D..U+10D21) placed after a consonant. These vowel signs represent short vowels. Vowel length and stress are indicated by three tone signs placed above vowel signs:

U+10D24 HANIFI ROHINGYA SIGN HARBAHAY

U+10D25 HANIFI ROHINGYA SIGN TAHALA

U+10D26 HANIFI ROHINGYA SIGN TANA

An independent or word-initial vowel is represented using the vowel carrier U+10D00 HANIFI ROHINGYA LETTER A plus a vowel sign.

Nasalization. Nasalization is indicated using the letter-like sign U+10D23 HANIFI ROHINGYA MARK NA KHONNA placed after a vowel sign.

Punctuation. Words are separated with spaces. Both Latin and Arabic punctuation signs such as periods, commas, colons, and hyphens are commonly used for delim-

iting text segments. An elongation feature using the Arabic *kashida* (U+0640 ARABIC TATWEEL) is often used for justification.

Digits. As in Arabic, digits in Hanifi Rohingya (U+1D030..U+1D039) are written from left to right.

Chapter 17

Southeast Asia-II

Indonesia and the Philippines

Four traditional Philippine scripts are described here: Tagalog (Baybayin), Hanunóo, Buhid, and Tagbanwa. They have limited current use. Each is a very simplified *abugida* which makes use of a few nonspacing vowel signs.

Although the official language of Indonesia, Bahasa Indonesia, is written in the Latin script, Indonesia has many local, traditional scripts, which are ultimately derived from Brahmi. Some of these scripts are documented in this chapter. Balinese and Javanese are closely related, highly ornate scripts; Balinese is primarily used for the Balinese language on the island of Bali, and Javanese for the Javanese language on the island of Java. Sundanese is used to write the Sundanese language on the island of Java. The Rejang script is used to write the Rejang language in southwest Sumatra, and the Batak script is used to write several Batak dialects, also on the island of Sumatra. Buginese (Lontara) and Makasar are two similar scripts that developed on the island of Sulawesi and are used to write Buginese, Makasar, and other languages.

Kawi, a historical script derived from Brahmi, is the common ancestor of several or perhaps all of the scripts described in this chapter. Kawi was used to write the Old Javanese, Sanskrit, Old Malay, Old Balinese, and Old Sundanese languages in insular southeast Asia between the 8th and 16th century.

17.1 Philippine Scripts: Tagalog, Hanunóo, Buhid, and Tagbanwa

17.1.1 Tagalog: U+1700–U+171F

Hanunóo: U+1720–U+173F

Buhid: U+1740–U+175F

Tagbanwa: U+1760–U+177F

The Tagalog (Baybayin), Hanunóo, Buhid, and Tagbanwa scripts are traditional scripts of the Philippines, and are in limited use today. South Indian scripts of the Pallava dynasty made their way to the Philippines, although the exact route is uncertain. They may have been transported by way of the Kavi scripts of Western Java between the tenth and fourteenth centuries CE.

Written accounts of the Tagalog script by Spanish missionaries and documents in Tagalog date from the mid-1500s. The first book in this script was printed in Manila in 1593. While the Tagalog script (also known as Baybayin), was used to write Tagalog, Bisaya, Ilocano, and other languages, it fell out of normal use by the mid-1700s. The modern Tagalog language (also known as Filipino) is now primarily written in the Latin script.

The Hanunóo, Buhid, and Tagbanwa scripts are related to Tagalog but may not be directly descended from it. The Hanunóo and the Buhid peoples live in Mindoro, while the Tagbanwa live in Palawan. Hanunóo enjoys the most use; it is widely used to write love poetry, a popular pastime among the Hanunóo. Tagbanwa is used less often.

17.1.2 Principles of the Philippine Scripts

The Philippine scripts share features with the other Brahmi-derived scripts to which they are related.

Consonant Letters. Philippine scripts have consonants containing an inherent *-a* vowel, which may be modified by the addition of vowel signs or canceled (killed) by the use of a virama-type mark. No conjunct consonants are employed in the Philippine scripts.

Two forms of the Tagalog letter *ra* are encoded: U+170D TAGALOG LETTER RA represents the preferred modern form that derived from the letter *da*. In contrast, U+171F

TAGALOG LETTER ARCHAIC RA represents a distinct historical form, also known as the Zambales *ra*.

Independent Vowel Letters. Philippine scripts use independent vowels to write syllables that do not begin with one of the consonant letters.

Dependent Vowel Signs. The vowel *-i* is written with a mark above the associated consonant, and the vowel *-u* with an identical mark below. The mark is known as *kudlit* "diacritic," *tuldik* "accent," or *tuldok* "dot" in Tagalog, and as *ulitan* "diacritic" in Tagbanwa. The Philippine scripts employ only the two vowel signs *i* and *u*, which are also used to stand for the vowels *e* and *o*, respectively.

Virama. Although all languages normally written with the Philippine scripts have syllables ending in consonants, not all of the scripts have a mechanism for expressing the canceled *-a*. As a result, in those orthographies, the final consonants are unexpressed.

Francisco Lopez introduced a cross-shaped *virama* for the Tagalog script in his 1620 catechism, but this innovation did not seem to find favor with native users, who seem to have considered the script adequate without it (they preferred ꚜꚜꙦ *kakapi* to ꚜꚜꙦ *kakampi*). A similar reform for the Hanunóo script seems to have been better received. The Hanunóo *pamudpod* was devised by Antoon Postma, who went to the Philippines from the Netherlands in the mid-1950s. In traditional orthography, ꙮ ꙰ꙮ꙰ ꙰ ꙰ꙮ꙰ *si apu ba upada* is, with the *pamudpod*, rendered more accurately as ꙮ ꙰ꙮ꙰ꙮ꙰ ꙰ꙮ꙰ ꙰ꙮ꙰ꙮ꙰ *si aypud bay upadan*; the Hanunóo pronunciation is *si aypod bay upadan*. U+1715 TAGALOG SIGN PAMUDPOD represents the *pamudpod* sign borrowed from Hanunóo for use in contemporary texts of the Tagalog script.

The Tagalog *virama*, Hanunóo *pamudpod*, and Tagalog *pamudpod* only cancel the inherent *-a*; they do not conjoin letters.

Directionality. The Philippine scripts are read from left to right in horizontal lines running from top to bottom. They may be written or carved either in that manner or in vertical lines running from bottom to top, moving from left to right. In the latter case, the letters are written sideways so they may be read horizontally. This method of writing is probably due to the medium and writing implements used. Text is often scratched with a sharp instrument onto beaten strips of bamboo, which are held pointing away from the body and worked from the proximal to distal ends, in columns from left to right.

Rendering. In Tagalog and Tagbanwa, the vowel signs simply rest over or under the consonants. In Hanunóo and Buhid, ligatures are often formed, as shown in *Table 17-1*.

Table 17-1. Hanunóo and Buhid Vowel Sign Combinations

Hanunóo			Buhid		
x	*x* + ̄	*x*+ ̥	*x*	*x* + ̄	*x*+ ̥
↗	↗̄	↗	⹀	⫞	☰
И	Ӣ	И	Ⴑ	Ⴄ	Ⴑ
K	K	K	ⴑ	ⴑ	ⴑ
W	W̄	W.	ᴡ	ᴡ	ᴡ
̌ƚ	̄̌ƚ	̌ƚ	̌ʋ	ʋ	̌ʋ
Ⴒ	Ⴒ̄	Ⴒ	Т	Ꞁ̄	Ꞁ
K	K	K	Ⴒ	Ⴒ	Ⴒ
7	7̄	7	Ꞁ	Ꞁ̄	Ꞁ
K	K̄	K	ⴷ	ⴷ	ⴷ
Ⴖ	Ⴖ̄	Ⴖ	Ⴖ	Ⴖ̄	Ⴖ
—	₣	₮	—	₣	₮
Ⴖ₁	Ⴖ̄₁	Ⴖ₁	ⴔ	ⴔ̄	ⴔ
ⱱ	ⱱ̄	ⱱ	ⴳ	ⴳ̄	ⴳ
Ⴖ₁	Ⴖ̄₁	Ⴖ₁	₸	₸̄	₸
ⱴ	ⱴ́	ⱴ	⌐	⌐	⌐

Punctuation. Punctuation has been unified for the Philippine scripts. In the Hanunóo block, U+1735 PHILIPPINE SINGLE PUNCTUATION and U+1736 PHILIPPINE DOUBLE PUNCTUATION are encoded.

17.2 Buginese

17.2.1 Buginese: U+1A00–U+1A1F

The Buginese script is used on the island of Sulawesi, mainly in the southwest. A variety of traditional literature has been printed in it. As of 1971, as many as 2.3 million speakers of Buginese were reported in the southern part of Sulawesi. The Buginese script is one of the easternmost of the Brahmi scripts and is perhaps related to Javanese. It is attested as early as the fourteenth century CE. Buginese bears some affinity to Tagalog and, like Tagalog, does not traditionally record final consonants. The Buginese language, an Austronesian language with a rich traditional literature, is one of the foremost languages of Indonesia. The script was previously also used to write the Makasar, Bimanese, and Madurese languages.

Repertoire. The repertoire contained in the Buginese block is intended to represent the core set of Buginese characters in standard printing fonts developed in the mid 19th century for the Bugis and Makasar languages. Variant letterforms and other extensions seen in palm leaf manuscripts or additional letters used in some languages are not yet encoded in this block. A visible virama symbol has also been attested, but is not needed for this core repertoire for Buginese.

Structure. Buginese vowel signs are used in a manner similar to that seen in other Brahmi-derived scripts. Consonants have an inherent /a/ vowel sound. Consonant conjuncts are not formed.

Ligature. One ligature is found in the Buginese script. It is formed by the ligation of <*a*, -*i*> + *ya* to represent *îya*, as shown in the first line of *Figure 17-1*. The ligature takes the shape of the Buginese letter *ya*, but with a dot applied at the far left side. Contrast that with the normal representation of the syllable *yi*, in which the dot indicating the vowel sign occurs in a centered position, as shown in the second line of *Figure 17-1*. The ligature for *îya* is not obligatory; it would be requested by inserting a *zero width joiner*.

Figure 17-1. Buginese Ligature
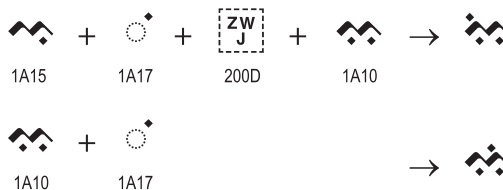
Order. Several orderings are possible for Buginese. The Unicode Standard encodes the Buginese characters in the Matthes order.

Punctuation. Buginese uses spaces between certain units. One punctuation symbol, U+1A1E BUGINESE PALLAWA, is functionally similar to the full stop and comma of the Latin script. There is also another separation mark, U+1A1F BUGINESE END OF SECTION.

U+A9CF JAVANESE PANGRANGKEP or a doubling of the vowel sign (especially U+1A19 BUGINESE VOWEL SIGN E and U+1A1A BUGINESE VOWEL SIGN O) is sometimes used to denote word reduplication. The shape of the Buginese reduplication sign is based on the Arabic digit two. The functionally similar U+A9CF JAVANESE PANGRANGKEP which has the same shape, is recommended for this sign in Buginese, rather than U+0662 ARABIC-INDIC DIGIT TWO, to avoid potential problems for text layout.

Numerals. There are no known digits specific to the Buginese script.

17.3 Balinese

17.3.1 Balinese: U+1B00–U+1B7F

The Balinese script, or *aksara Bali*, is used for writing the Balinese language, the native language of the people of Bali, known locally as *basa Bali*. It is a descendant of the ancient Brahmi script of India, and therefore it has many similarities with modern scripts of South Asia and Southeast Asia, which are also members of that family. The Balinese script is used to write Kawi, or Old Javanese, which strongly influenced the Balinese language in the eleventh century CE. The script is also used to write the Sasak language, which is spoken on the island of Lombok to the east of Bali. Some Balinese words have been borrowed from Sanskrit, which may also be written in the Balinese script.

Structure. Balinese consonants have an inherent *-a* vowel sound. Consonants combine with following consonants in the usual Brahmic fashion: the inherent vowel is "killed" by U+1B44 ◌ BALINESE ADEG ADEG (*virama*), and the following consonant is subjoined, often with a change in shape. *Table 17-2* shows the base consonants and their conjunct forms.

Table 17-2. Balinese Base Consonants and Conjunct Forms

Consonant	Base Form	Conjunct Form
ka	ᬓ	◌
kha	ᬔ	◌
ga	ᬕ	◌
gha	ᬖ	◌
nga	ᬗ	◌
ca	ᬘ	◌
cha	ᬙ	◌
ja	ᬚ	◌
jha	ᬛ	◌
nya	ᬜ	◌
tta	ᬝ	◌

ttha	ᬝ	◌ᬝ
dda	ᬟ	◌ᬟ
ddha	ᬠ	◌ᬠ
nna	ᬡ	◌ᬡ
ta	ᬢ	◌ᬢ
tha	ᬣ	◌ᬣ
da	ᬤ	◌ᬤ
dha	ᬥ	◌ᬥ
na	ᬦ	◌ᬦ
pa	ᬧ	◌ᬧ
pha	ᬨ	◌ᬨ
ba	ᬩ	◌ᬩ
bha	ᬪ	◌ᬪ
ma	ᬫ	◌ᬫ
ya	ᬬ	◌ᬬ
ra	ᬭ	◌ᬭ
la	ᬮ	◌ᬮ
wa	ᬯ	◌ᬯ
sha	ᬰ	◌ᬰ
ssa	ᬱ	◌ᬱ
sa	ᬲ	◌ᬲ
ha	ᬳ	◌ᬳ

The seven letters U+1B45 ᭅ BALINESE LETTER KAF SASAK through U+1B4B ᭋ
BALINESE LETTER ASYURA SASAK were proposed in the late 20th century as exten-
sions for the Sasak language to replace use of the nukta, U+1B34 ◌᬴ BALINESE SIGN
REREKAN, but have seen little use.

Balinese dependent vowel signs are used in a manner similar to that employed by other Brahmic scripts.

Independent vowels are used in a manner similar to that seen in other Brahmic scripts, with a few differences. For example, U+1B05 ᬅ BALINESE LETTER AKARA and U+1B0B ᬋ BALINESE LETTER RA REPA can be treated as consonants; that is, they can be followed by *adeg adeg*. In Sasak, the vowel letter *akara* can be followed by an explicit *adeg adeg* ᬅ᭄ in word- or syllable-final position, where it indicates the glottal stop; other consonants can also be subjoined to it.

Behavior of ra. U+1B03 ᬃ BALINESE SIGN SURANG typically represents a final consonant -*r*. This sign is derived from the cluster-initial sign *r-* (also known as *repha*) of the parent script Kawi; it still represents a *repha* when transliterating Kawi, but it has been reanalyzed to represent a final -*r* in the Balinese orthography. As shown in <u>Figure 17-2</u>, the same written form, pronounced as *dhamar* in the Balinese language, represents *dharma* in transliterated Kawi. Because a *surang* used as a final -*r* cannot be visually distinguished from a *surang* used as *repha*, they are encoded in the same way. When combined with another above-base sign, a *surang* used as *repha* may be rendered to the left of the other sign rather than to the right.

Figure 17-2. Writing *dharma* in Balinese
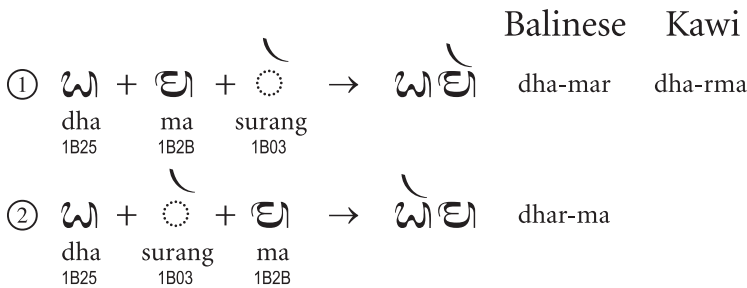
For searching and sorting, *surang* should be treated as equivalent to *ra*. When the processed text is transliterated Kawi, *surang* also needs to be reordered to precede its orthographic syllable. Two other combining signs are also equivalent to base letters for searching and sorting: U+1B02 ᬂ BALINESE SIGN CECEK (*anusvara*) is equivalent to *nga*, and U+1B04 ᬄ BALINESE SIGN BISAH (*visarga*) is equivalent to *ha*.

Behavior of ra repa. The unique behavior of U+1B0B ᬋ BALINESE LETTER RA REPA (*vocalic r*) results from a reanalysis of the independent vowel letter as a consonant. In a compound word in which the first element ends in a consonant and the second element begins with an original *ra* + *pepet*, such as *Pak Rërëh* ᬧᬓᬺᬭᭂᬄ "Mr Rërëh", the subjoined form of ᬺ *ra repa* is used; this particular sequence is encoded *ka* + *adeg adeg* + *ra repa*. However, in other contexts where the *ra repa* represents the original Sanskrit vowel, U+1B3A ᬺ BALINESE VOWEL SIGN RA REPA is used, as in *Krësna* ᬓᬺᬱ᭄ᬡ.

Rendering. The vowel signs *u* and *uu* take different forms when combined with subscripted consonant clusters, as shown in *Table 17-3*. The upper limit of consonants in a cluster is three, the last of which can be *y*, *w*, or *r*.

Table 17-3. Balinese Consonant Clusters with *u* and *uu*

Syllable	Glyph
kyu	ᬓ᭄ᬬᬸ
kyuu	ᬓ᭄ᬬᬹ
kwu	ᬓ᭄ᬯᬸ
kwuu	ᬓ᭄ᬯᬹ
kru	ᬓ᭄ᬭᬸ
kruu	ᬓ᭄ᬭᬹ
kryu	ᬓ᭄ᬭ᭄ᬬᬸ
kryuu	ᬓ᭄ᬭ᭄ᬬᬹ
skru	ᬲ᭄ᬓ᭄ᬭᬸ
skruu	ᬲ᭄ᬓ᭄ᬭᬹ

Nukta. The combining mark U+1B34 ◌᬴ BALINESE SIGN REREKAN (*nukta*) is used to extend the character repertoire for foreign sounds.

Archaic Jnya. The character U+1B4C ᭌ BALINESE LETTER ARCHAIC JNYA is occasionally used in older texts in place of *ja* +subjoined *nya*. Both forms may be present in the same text, but the archaic form is not found in modern Balinese texts. A subjoined form of this character is unattested.

Ordering. The traditional order *ha na ca ra ka | da ta sa wa la | ma ga ba nga | pa ja ya nya* is taught in schools, although van der Tuuk followed the Javanese order *pa ja ya nya | ma ga ba nga* for the second half. The arrangement of characters in the code charts follows the Brahmic ordering.

Punctuation. U+1B5E ᭞ BALINESE CARIK SIKI and U+1B5F ᭟ BALINESE CARIK PAREREN are used as comma and full stop, respectively. Their inverted versions U+1B4E ᭎ BALINESE INVERTED CARIK SIKI and U+1B4F ᭏ BALINESE INVERTED CARIK PAREREN have been used in some manuscripts to indicate finer subdivisions. U+1B5D ᭝ BALINESE CARIK PAMUNGKAH is used as a colon.

Both U+1B5A ꭰ BALINESE PANTI and U+1B5B ꭱ BALINESE PAMADA are used to begin a section of text. A shorter version of *panti*, U+1B7F ꞈ BALINESE PANTI BAWAK, may be used to indicate finer subdivisions.

A variety of punctuation marks are used to indicate the end of a section. These usually consist of U+1B5C ꭲ BALINESE WINDU enclosed within two other punctuation marks, which vary depending on which sign began the section. Examples include: *carik siki* ꭳ, *carik pareren* ꭴ (sometimes called *pasalinan*), *panti* ꭵ, and *carik agung* ꭶ.

At the end of a text, U+1B7D ꭷ BALINESE PANTI LANTANG and U+1B7E ꭸ BALINESE PAMADA LANTANG may be used, depending on the secular or religious nature of the text. These may also be used together with U+1B5C ꭲ BALINESE WINDU or their short counterparts in combinations such as ꭹ and ꭺ.

Line Breaking. Line breaks may occur after any orthographic syllable. Traditional Balinese texts are written on palm leaves; books of these leaves bound together are called *lontar*. U+1B60 ꭻ BALINESE PAMENENG may be inserted in lontar texts at the end of a line to fill the line.

Musical Symbols. Bali is well known for its rich musical heritage. A number of related notation systems are used to write music. To represent degrees of a scale, the syllables *ding dong dang deng dung* are used (encoded at U+1B61..U+1B64, U+1B66), in the same way that *do re mi fa so la ti* is used in Western tradition. The symbols representing these syllables are based on the vowel matras, together with some other symbols. However, unlike the regular vowel matras, these stand-alone spacing characters take diacritical marks. They also have different positions and sizes relative to the baseline. These matra-like symbols are encoded in the range U+1B61..U+1B6A, along with a modified *aikara*. Some notation systems use other spacing letters, such as U+1B09 ꬵ BALINESE LETTER UKARA and U+1B27 ꬷ BALINESE LETTER PA, which are not separately encoded for musical use. The U+1B01 ꬀ BALINESE SIGN ULU CANDRA (*candrabindu*) can also be used with U+1B62 ꭼ BALINESE MUSICAL SYMBOL DENG and U+1B68 ꭽ BALINESE MUSICAL SYMBOL DEUNG, and possibly others. BALINESE SIGN ULU CANDRA can be used to indicate modre symbols as well.

A range of diacritical marks is used with these musical notation base characters to indicate metrical information. Some additional combining marks indicate the instruments used; this set is encoded at U+1B6B..U+1B73. A set of symbols describing certain features of performance are encoded at U+1B74..U+1B7C. These symbols describe the use of the right or left hand, the open or closed hand position, the "male" or "female" drum (of the pair) which is struck, and the quality of the striking.

More information about Balinese musical notations is available in Unicode Technical Note 51, "Musical Symbols and Sasak Characters in the Balinese Script".

Modre Symbols. The Balinese script also includes a range of "holy letters" called modre symbols. Most of these letters can be composed from the constituent parts currently encoded, including U+1B01 ◌̍ BALINESE SIGN ULU CANDRA.

17.4 Javanese

17.4.1 Javanese: U+A980–U+A9DF

The Javanese script, or *aksara Jawa*, is used for writing the Javanese language, known locally as *basa Jawa*. The script is a descendent of the ancient Brahmi script of India, and so has many similarities with the modern scripts of South Asia and Southeast Asia which are also members of that family. The Javanese script is also used for writing Sanskrit, Jawa Kuna (a kind of Sanskritized Javanese), and transcriptions of Kawi, as well as the Sundanese language, also spoken on the island of Java, and the Sasak language, spoken on the island of Lombok.

The Javanese script was in current use in Java until about 1945; in 1928 Bahasa Indonesia was made the national language of Indonesia and its influence eclipsed that of other languages and their scripts. Traditional Javanese texts are written on palm leaves; books of these bound together are called *lontar*, a word which derives from ron "leaf" and tal "palm".

Consonants. Consonants have an inherent *-a* vowel sound. Consonants combine with following consonants in the usual Brahmic fashion: the inherent vowel is "killed" by U+A9C0 JAVANESE PANGKON, and the following consonant is subjoined, often with a change in shape.

In Javanese, Sanskrit vocalic liquids (short and long versions of *ṛ* and *ḷ*) are treated as consonant letters with an alternate inherent vowel: *rĕ*, *reu*, *lĕ*, and *leu*; they are not independent vowels with dependent vowel equivalents, as is the case in Balinese or Devanagari. Short and long versions of the *vocalic-ḷ* are separately encoded, as U+A98A JAVANESE LETTER NGA LELET and U+A98B JAVANESE LETTER NGA LELET RASWADI. In contrast, the long version of the *vocalic-ṛ* is represented by a sequence of the short vowel U+A989 JAVANESE LETTER PA CEREK followed by the dependent vowel sign *-aa*, U+A9B4 JAVANESE VOWEL SIGN TARUNG, serving as a length mark in this case.

U+A9B3 JAVANESE SIGN CECAK TELU is a diacritic used with various consonantal base letters to represent foreign sounds. Typically these diacritic-marked consonants are used for sounds borrowed from Arabic.

Independent Vowels. Independent vowel letters are used essentially as in other Brahmic scripts. Modern Javanese uses U+A986 JAVANESE LETTER I and U+A987 JAVANESE LETTER II for short and long *i*, but the Kawi orthography instead uses U+A985 JAVANESE LETTER I KAWI and U+A986 JAVANESE LETTER I for short and long *i*, respectively.

The long versions of the *u* and *o* vowels are written as sequences, using U+A9B4 JAVANESE VOWEL SIGN TARUNG as a length mark.

Dependent Vowels. Javanese—unlike Balinese—represents multipart dependent vowels with sequences of characters, in a manner similar to the Myanmar script. The Balinese community considers it important to be able to directly transliterate Sanskrit into Balinese, so multipart dependent vowels are encoded as single, composite forms in Balinese, as is done in Devanagari. In contrast, for the Javanese script, the correspondence with Sanskrit letters is not so critical, and a different approach to the encoding has been taken. Similar to the treatment of long versions of Javanese independent vowels, the two-part dependent vowels are explicitly represented with a sequence of two characters, using U+A9B4 JAVANESE VOWEL SIGN TARUNG, as shown in *Figure 17-3*.

Figure 17-3. Representation of Javanese Two-Part Vowels
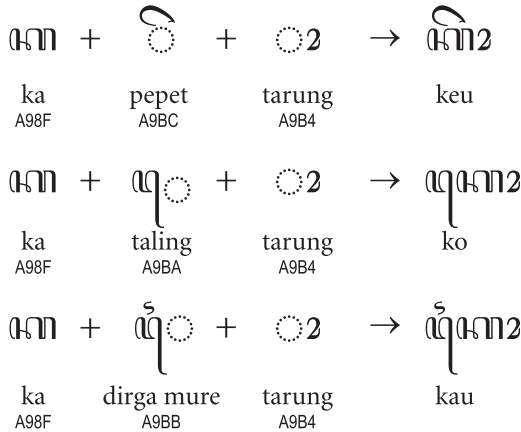
Tarung is not used alone when writing the Javanese language, but it represents the vowel *aa* when writing Sanskrit and *o* when writing Sundanese. An alternative glyph of *tarung* has been separately encoded as U+A9B5 JAVANESE VOWEL SIGN TOLONG, which is not normally needed, except when used in contrast with the ordinary *tarung*.

Consonant Signs. The characters U+A980 JAVANESE SIGN PANYANGGA, U+A981 JAVANESE SIGN CECAK, and U+A983 JAVANESE SIGN WIGNYAN are analogous to U+0901 DEVANAGARI SIGN CANDRABINDU, U+0902 DEVANAGARI SIGN ANUSVARA, and U+0903 DEVANAGARI SIGN VISARGA and behave in much the same way.

There are three medial consonant signs, U+A9BD JAVANESE CONSONANT SIGN KERET, U+A9BE JAVANESE CONSONANT SIGN PENGKAL, and U+A9BF JAVANESE CONSONANT SIGN CAKRA, which represent *-rĕ, -ya,* and *-ra* respectively. These medial consonant signs contrast with the subjoined forms of the letters *rĕ*, *ya*, and *ra*. The

subjoined forms may indicate a syllabic boundary, whereas *keret*, *pengkal*, and *cakra* are used in ordinary consonant clusters.

Rendering. There are many conjunct forms in Javanese, though most are fairly regular and easy to identify. Subjoined consonants and vowel signs rendered below them usually interact typographically. For example, the vowel signs [u] and [u:] take different forms when combined with subscripted consonant clusters. Consonant clusters may have up to three elements. In three-element clusters, the last element is always one of the medial glides: *-ya*, *-wa*, or *-ra*.

Digits. The Javanese script has its own set of digits, seven of which (1, 2, 3, 6, 7, 8, 9) look just like letters of the alphabet. Implementations with concerns about security issues need to take this into account. The punctuation mark U+A9C7 JAVANESE PADA PANGKAT is often used with digits in order to help to distinguish numbers from sequences of letters.

Punctuation. A large number of punctuation marks are used in Javanese. Titles may be flanked by the pair of ornamental characters, U+A9C1 JAVANESE LEFT RERENGGAN and U+A9C2 JAVANESE RIGHT RERENGGAN; glyphs used for these may vary widely.

U+A9C8 JAVANESE PADA LINGSA is a danda mark that corresponds functionally to the use of a comma. The doubled form, U+A9C9 JAVANESE PADA LUNGSI, corresponds functionally to the use of a full stop. It is also used as a "ditto" mark in vertical lists. U+A9C7 JAVANESE PADA PANGKAT is used much like the European colon.

U+A9C7 JAVANESE PADA PANGKAT is used to abbreviate personal names and is placed at the end of the abbreviation.

The doubled U+A9CB JAVANESE PADA ADEG ADEG typically begins a paragraph or section, while the simple U+A9CA JAVANESE PADA ADEG is used as a common divider though it can be used in pairs marking text for attention. The two characters, U+A9CC JAVANESE PADA PISELEH and U+A9CD JAVANESE TURNED PADA PISELEH, are used similarly, either both together or with U+A9CC JAVANESE PADA PISELEH simply repeated.

The punctuation ring, U+A9C6 JAVANESE PADA WINDU, is not used alone, a situation similar to the pattern of use for its Balinese counterpart U+1B5C BALINESE WINDU. When used with U+A9CB JAVANESE PADA ADEG ADEG this *windu* sign is called *pada guru*, *pada bab*, or *uger-uger*, and is used to begin correspondence where the writer does not desire to indicate a rank distinction as compared to his audience. More formal letters may begin with one of the three signs: U+A9C3 JAVANESE PADA ANDAP (for addressing a higher-ranked person), U+A9C4 JAVANESE PADA MADYA (for addressing an equally-ranked person), or U+A9C5 JAVANESE PADA LUHUR (for addressing a lower-ranked person).

Reduplication. U+A9CF JAVANESE PANGRANGKEP is used to show the reduplication of a syllable. The character derives from U+0662 ARABIC-INDIC DIGIT TWO but in Javanese it does not have a numeric use. The Javanese reduplication mark is encoded as a separate character from the Arabic digit, because it differs in its Bidi_Class property value.

Line Breaking. Opportunities for line breaking occur after any full orthographic syllable. Hyphens are not used.

In some printed texts, an epenthetic spacing U+A9BA JAVANESE VOWEL SIGN TALING is placed at the end of a line when the next line begins with the glyph for U+A9BA JAVANESE VOWEL SIGN TALING, which is reminiscent of a specialized hyphenation (or of quire marking). This practice is nearly impossible to implement in a free-flowing text environment. Typographers wishing to duplicate a printed page may manually insert U+00A0 NO-BREAK SPACE before U+A9BA JAVANESE VOWEL SIGN TALING at the end of a line, but this would not be orthographically correct.

17.5 Rejang

17.5.1 Rejang: U+A930–U+A95F

The Rejang language is spoken by about 200,000 people living on the Indonesian island of Sumatra, mainly in the southwest. There are five major dialects: Lebong, Musi, Kebanagun, Pesisir (all in Bengkulu Province), and Rawas (in South Sumatra Province). Most Rejang speakers live in fairly remote rural areas, and slightly less than half of them are literate.

The Rejang script was in use prior to the introduction of Islam to the Rejang area. The earliest attested document appears to date from the mid-18th century CE. The traditional Rejang corpus consists chiefly of ritual texts, medical incantations, and poetry.

Structure. Rejang is a Brahmi-derived script. It is related to other scripts of the Indonesian region, such as Batak and Buginese.

Consonants in Rejang have an inherent /a/ vowel sound. Vowel signs are used in a manner similar to that employed by other Brahmi-derived scripts. There are no consonant conjuncts. The basic syllabic structure is C(V)(F): a consonant, followed by an optional vowel sign and an optional final consonant sign or virama.

Rendering. Rejang texts tend to have a slanted appearance typified by the appearance of U+A937 REJANG LETTER BA. This sense that the script is tilted to the right affects the placement of the combining marks for vowel signs. Vowel signs above a letter are offset to the right, and vowel signs below a letter are offset to the left, as the "above" and "below" positions for letters are perceived in terms of the overall slant of the letters.

Ordering. The ordering of the consonants and vowel signs for Rejang in the code charts follows a generic Brahmic script pattern. The Brahmic ordering of Rejang consonants is attested in numerous sources. There is little evidence one way or the other for preferences in the relative order of Rejang vowel signs and consonant signs.

Digits. There are no known script-specific digits for the Rejang script.

Punctuation. European punctuation marks such as comma, full stop, and colon, are used in modern writing. U+A95F REJANG SECTION MARK may be used at the beginning and end of paragraphs.

Traditional Rejang texts tend not to use spaces between words, but their use does occur in more recent texts. There is no known use of hyphenation.

17.6 Batak

17.6.1 Batak: U+1BC0–U+1BFF

The Batak script is used on the island of Sumatra to write the five Batak dialects: Karo, Mandailing, Pakpak, Simalungun, and Toba. The script is called *si-sia-sia* or *surat na sampulu sia*, which means "the nineteen letters." The script is taught in schools mainly for cultural purposes, and is used on some signs for shops and government offices.

Structure. Batak is a Brahmi-derived script. It is written left to right.

Consonants in Batak have an inherent /a/ vowel sound. Batak uses a vowel killer which is called *pangolat* in Mandailing, Pakpak, and Toba. In Karo the killer is called *penengen*, and in Simalungen it is known as *panongonan*. The appearance of the killer differs between some of the dialects.

Batak has three independent vowels and makes use of a number of vowel signs and two consonant signs. Some vowel signs are only used by certain language communities. There are no consonant conjuncts. The basic syllabic structure is $C(V)(C_s|C_d)$: a consonant, followed by an optional vowel sign, which may be followed either by a consonant sign C_s (*-ng* or *-h*) or a killed final consonant C_d.

Rendering. Most vowel signs and the two killers, U+1BF2 BATAK PANGOLAT and U+1BF3 BATAK PANONGONAN, are spacing marks. U+1BEE BATAK VOWEL SIGN U can ligate with its base consonant.

The two consonant signs, U+1BF0 BATAK CONSONANT SIGN NG and U+1BF1 BATAK CONSONANT SIGN H, are nonspacing marks, usually rendered above the spacing vowel signs. When U+1BF0 BATAK CONSONANT SIGN NG occurs together with the nonspacing mark, U+1BE9 BATAK VOWEL SIGN EE, both are rendered above the base consonant, with the glyph for the *ee* at the top left and the glyph for the *ng* at the top right.

The main peculiarity of Batak rendering concerns the reordering of the glyphs for vowel signs when one of the two killers, *pangolat* or *panongonan*, is used to close the syllable by killing the inherent vowel of a final consonant. This reordering for display is entirely regular. So, while the representation of the syllable /tip/ is done in logical order: <*ta*, *vowel sign i*, *pa*, *pangolat*>, when rendered for display the glyph for the vowel sign is visually applied to the final consonant, *pa*, rather than to the *ta*. The glyph for the *pangolat* always stays at the end of the syllable.

Punctuation. Punctuation is not normally used; instead all letters simply run together. However, a number of *bindu* characters are occasionally used to disam-

biguate similar words or phrases. U+1BFF BATAK SYMBOL BINDU PANGOLAT is trailing punctuation, following a word, surrounding the previous character somewhat.

The minor mark used to begin paragraphs and stanzas is U+1BFC BATAK SYMBOL BINDU NA METEK, which means "small bindu." It has a shape-based variant, U+1BFD BATAK SYMBOL BINDU PINARBORAS ("rice-shaped bindu"), which is likewise used to separate sections of text. U+1BFE BATAK SYMBOL BINDU JUDUL ("title bindu") is sometimes used to separate a title from the main text, which normally begins on the same line.

Line Breaking. Traditionally, line breaks can occur before any spacing character. However, the vowel reordering described above is required even when a line break occurs between the characters involved. In typical Unicode-based implementations, this requires keeping the characters involved on the same line.

17.7 Sundanese

17.7.1 Sundanese: U+1B80–U+1BBF

The Sundanese script, or *aksara Sunda*, is used for writing the Sundanese language, one of the languages of the island of Java in Indonesia. It is a descendant of the ancient Brahmi script of India, and so has similarities with the modern scripts of South Asia and Southeast Asia which are also members of that family. The script has official support. It is taught in schools and used on road signs.

The Sundanese language has been written using a number of different scripts over the years. Pallawa or Pra-Nagari was first used in West Java to write Sanskrit from the fifth to the eighth centuries CE. *Sunda Kuna* or Old Sundanese was derived from Pallawa and was used in the Sunda Kingdom from the 14th to the 18th centuries. The earliest example of Old Sundanese is the Prasasti Kawali stone. The Javanese script was used to write Sundanese from the 17th to the 19th centuries, and the Arabic script was used from the 17th to the 20th centuries. The Latin script has been in wide use since the 20th century. The modern Sundanese script, called S*unda Baku* or Official Sundanese, became official in 1996. This modern script was derived from Old Sundanese.

Structure. Sundanese consonants have an inherent vowel /a/. This inherent vowel can be modified by the addition of dependent vowel signs (*matras*). The script also has independent vowels.

In the modern orthography, an explicit vowel killer character, U+1BAA SUNDANESE SIGN PAMAAEH, is used to indicate the absence, or "killing," of the inherent vowel, but does not build consonant conjuncts. In Old Sundanese, however, consonant conjuncts do appear, and are formed with U+1BAB SUNDANESE SIGN VIRAMA.

Medials. In the modern orthography, initial Sundanese consonants can be followed by one of the three consonant signs for medial consonants, *-ya*, *-ra*, or *-la*. These medial consonants are graphically displayed as subjoined elements to their base consonants, and are not considered conjuncts proper, because they are not formed using a *virama*. In Old Sundanese, a subjoined *ma*, U+1BAC SUNDANESE CONSO-NANT SIGN PASANGAN MA, and a subjoined *wa*, U+1BAD SUNDANESE CONSONANT SIGN PASANGAN WA, occur. They contrast with the conjunct forms created with the *virama*.

Final Consonants. Sundanese historical texts employ a final consonant, U+1BBE 𝼮 SUNDANESE LETTER FINAL K, which is distinct from the modern representation with the explicit vowel killer U+1BAA ◌ SUNDANESE SIGN PAMAAEH: 𝼮 <1B8A, 1BAA>. U+1BBF 𝼯 SUNDANESE LETTER FINAL M was used in a 21st-century document, based on a scribal error in an old Sundanese manuscript, and should not be used in

current practice. Rather, both old and modern representations of *final m* use ꦹ₃ <1B99, 1BAA>.

Combining Marks. Three final consonants are separately encoded as combining marks: *-ng, -r, -h*. These are analogues of Brahmic *anusvara*, *repha*, and *visarga*, respectively.

Historic Characters. Additional historic consonants appear only in old texts: *reu*, *leu*, and *archaic i*. The *archaic i* is represented by U+1BBD ꮝ SUNDANESE LETTER BHA because it was misinterpreted as *bha* in early transcriptions; the erroneous name has been corrected with formal name alias SUNDANESE LETTER ARCHAIC I.

Another historic character, U+1BBA ꮊ SUNDANESE AVAGRAHA, has two functions. In one, it kills the inherent vowel of the preceding consonant and causes hiatus before an initial *a*. In the other, it doubles the preceding consonant, from which it may be separated in writing by a dependent vowel.

Additional Consonants. Two supplemental consonant letters are used in the modern script: U+1BAE SUNDANESE LETTER KHA and U+1BAF SUNDANESE LETTER SYA. These are used to represent the borrowed sounds denoted by the Arabic letters *kha* and *sheen*, respectively.

Digits. Sundanese has its own script-specific digits, which are separately encoded in this block.

Punctuation. Sundanese uses European punctuation marks, such as comma, full stop, question mark, and quotation marks. Spaces are used in text. Opportunities for hyphenation occur after any full orthographic syllable.

Ordering. The order of characters in the code charts follows the Brahmic ordering. The *ha-na-ca-ra-ka* order found in Javanese and Balinese does not seem to be used in Sundanese.

Ordering of Syllable Components. Dependent vowels and other signs are encoded after the consonant to which they apply. The ordering of elements for the modern Sundanese orthography is shown in more detail in *Table 17-4*.

Table 17-4. Modern Sundanese Syllabic Structure

Class	Examples	Encoding
consonant or independent vowel	ꦴ	[U+1B83..U+1BA0, U+1BAE, U+1BAF]
consonant sign -ya, -ra, -la	ꦴ,ꦴ,ꦴ	[U+1BA1..U+1BA3]
dependent vowel, killer	ꦴ,ꦴ₃	[U+1BA4..U+1BA9, U+1BAA]
final consonant	ꦴ	[U+1B80..U+1B82]

The killer (*pamaaeh*) occupies the same logical position as a dependent vowel, but indicates the absence, rather than the presence of a vowel. It cannot be followed by a combining mark for a final consonant, nor can it be preceded by a consonant sign.

The left-side dependent vowel U+1BA6 SUNDANESE VOWEL SIGN PANAELAENG occurs in logical order after the consonant (and any medial consonant sign), but in visual presentation its glyph appears *before* (to the left of) the consonant.

Rendering. When more than one sign appears above or below a consonant, the two are rendered side-by-side, rather than being stacked vertically.

17.7.2 Sundanese Supplement: U+1CC0–U+1CCF

The Sundanese Supplement block contains eight *bindu* punctuation marks found in historical materials.

17.8 Makasar

17.8.1 Makasar: U+11EE0–U+11EFF

The Makasar script was used historically in South Sulawesi, Indonesia for writing the Makasar language. It is sometimes spelled "Makassar," and is also referred to as "Old Makassarese" or "Makassarese bird script." The script was maintained for official purposes in the kingdoms of Makasar in the 17th century, and it was used for writing a number of historical accounts, such as the "Chronicles of Gowa and Tallo'," but it was superseded by the Buginese script in the 19th century and is no longer used. Although Makasar is thought to have evolved from Rejang, it shares several similarities with Buginese.

Structure. Makasar is a Brahmi-derived *abugida*. It is written horizontally, from left to right. Consonant signs carry an inherent /a/ vowel sign. Alternative vowel sounds are expressed by applying one of four combining characters to a consonant. Each vowel sign appears on a different side of the base consonant: right, left, top, and bottom. They are all encoded as combining characters following the consonant.

Like Buginese, geminated and clustered consonants are not indicated, nor are syllable-final consonants. However, Makasar differs from the Buginese script in that it does not have the pre-nasalized clusters, such as /ŋka/, that occur in Buginese, and it includes special features for consonant repetition.

There is only one independent vowel sign, U+11EF1 ꜫ MAKASAR LETTER A. Vowel signs can be attached to this character to produce other vowel sounds when a syllable has no consonant, such as at the beginning of a word.

Consonant Repetition. Adjacent syllables that use the same consonant can be written by appending two vowel signs to a single consonant, as shown in the following example. Usually both vowels are the same in this case, and a consonant can take a maximum of two vowel signs.

    U+11EE7 ꜧ *da* + U+11EF4 ◌ *vowel sign u* + U+11EF4 ◌ *vowel sign u* → ꜧ [dudu]

U+11EF2 Ꜳ MAKASAR ANGKA can also be used to repeat the consonant used in the previous syllable. This is particularly useful when one or both syllables use the inherent vowel, but *angka* may also be followed by a different vowel sound from that of the previous syllable. *Angka* is associated with the inherent vowel or a vowel sign in the same way as any normal consonant character. For example:

    U+11EED ꜭ *ra* + U+11EF4 ◌ *vowel sign u* + U+11EF2 Ꜳ *angka* → ꜭꜲ [rura]

    U+11EE5 ꜥ *ma* + U+11EF2 Ꜳ *angka* + U+11EF3 ◌ *vowel sign i* → ꜥꜲ [mami]

Letter va. U+11EEF MAKASAR LETTER VA is named "VA" even though the consonant is pronounced /w/ in the Makasar language. The name for this character aligns with the name for the related letter U+1A13 BUGINESE LETTER VA.

Digits. The available Makasar manuscript sources show two distinct sets of digits. The first set strongly resembles European digits and can be represented with U+0030..U+0039. The second set strongly resembles Arabic-Indic digits, and can be represented with U+0660..U+0669. Therefore, script-specific digits for Makasar are not separately encoded. Digits are frequently used, and both sets occur concurrently in the sources.

The Arabic-Indic digits are restricted to Arabic-language environments—particularly for expressing dates of the Hijri era. The European digits are used for general purposes, but occur within Arabic-language contexts for writing non-Hijri dates, specifically those of the Gregorian calendar.

Digits may occur above U+0600 ؀ ARABIC NUMBER SIGN or U+0601 ؁ ARABIC SIGN SANAH, see *Figure 9-7* for an example.

Punctuation. Sentences are delimited with U+11EF7 ⸮ MAKASAR PASSIMBANG, and sections are terminated with U+11EF8 ⸽ MAKASAR END OF SECTION. Words are often, but not always, separated by spaces. Line breaks normally appear after syllable boundaries. Hyphens or other marks indicating continuance are not used.

The end of a text is often marked using a stylized rendering of the Arabic word *tammat* تمّت, meaning "it is complete." There is no atomic character encoded for this symbol, so the sequence should be represented using Arabic letters <*ta* + *meem* + *shadda* + *ta*>, where the *shadda* is optional.

17.9 Kawi

17.9.1 Kawi: U+11F00–U+11F5F

The Kawi script is a historical Brahmi-derived script that was used between the 8th and 16th century in insular southeast Asia to write the Old Javanese, Sanskrit, Old Malay, Old Balinese, and Old Sundanese languages. A large portion of its corpus is found in Java, but Kawi materials have also been found in Sumatra, the Malay Peninsula, Bali, and the Philippines. Letter shapes evolved significantly over its 800 years of use, and later Kawi shows many variations over its wide geographic distribution; eventually, these variants evolved into the many modern Brahmic scripts of insular southeast Asia. The 21st century has brought renewed interest in the script, including some use in social media to write the modern Javanese or Indonesian languages.

The typeface used here is primarily based on early Kawi inscriptions, with some glyphs adapted from later attestations.

Structure. The Kawi script is an *abugida* and written from left to right. The inherent vowel of a consonant can be overridden by attaching a dependent vowel sign. It can also be suppressed by attaching a *virama* sign, U+11F41 ◌ KAWI SIGN KILLER, or the conjunct form of another consonant or vocalic liquid, which is encoded by preceding the consonant or vocalic liquid with U+11F42 ◌ KAWI CONJOINER. A vowelless consonant *r-* that starts an orthographic syllable may be represented by a *repha*, which is encoded as U+11F02 ◌ KAWI SIGN REPHA. Consonant stacks with up to four consonants are known.

Consonants. *Table 17-5* shows the base consonants and their conjunct forms.

Table 17-5. Kawi Base Consonants and Conjunct Forms

Consonant	Base Form	Conjunct Form
ka	ᰀ	ᰀ
kha	ᰁ	ᰁ
ga	ᰂ	ᰂ
gha	ᰃ	ᰃ
nga	ᰄ	ᰄ
ca	ᰅ	ᰅ
cha	ᰆ	ᰆ
ja	ᰇ	ᰇ
jha	ᰈ	ᰈ

nya	ᨎ	◌ᩲ
tta	ᨈ	◌ᨭ
ttha	ᨈ	◌ᨮ
dda	ᨯ	◌ᨯ
ddha	ᨯ	◌ᨰ
nna	ᨏ	◌ᨱ
ta	ᨲ	◌ᨲ
tha	ᨳ	◌ᨳ
da	ᨴ	◌ᨴ
dha	ᨵ	◌ᨵ
na	ᨶ	◌ᨶ
pa	ᨹ	◌ᨸ
pha	ᨺ	◌ᨹ
ba	ᨻ	◌ᨻ
bha	ᨽ	◌ᨽ
ma	ᨾ	◌ᨾ
ya	ᨿ	ᨿ
ra	ᩁ	◌, ◌ᩁ
la	ᩃ	◌ᩃ
wa	ᩅ	◌ᩅ
sha	ᩆ	◌ᩆ
ssa	ᩇ	◌ᨸ, ◌
sa	ᩈ	◌ᩈ
ha	ᩉ	◌ᩉ, ◌ᩌ

The below-base conjunct form of *ra* is commonly used when the pre-base form would collide with other marks, but can also be used as a stylistic variant. The second conjunct forms of *ssa* and *ha* are stylistic variants.

A vowelless *r-* that starts an orthographic syllable is normally written with a *repha* above the following consonant, but occasionally with the base form of *ra* with a subjoined consonant, for example, *rwa* ᩁ᩠ᩅ versus ᩁ. In some late Kawi varieties, the *repha* glyph may be used for a final *-r* consonant.

U+11F33 ꜱ KAWI LETTER JNYA is a graphic simplification of the consonant cluster ꜳ *jnya*; it has no conjunct form. Additional marks can be attached to it.

Independent Vowels. The Kawi script has a set of independent vowels and vocalic liquid letters. Dependent vowel signs and other signs can be attached to them. Letters *au*, *eu*, and *euu* are visually composites of other letters and dependent vowels, and are encoded as such. Letters *aa*, *ii*, and *uu* occur in both composite and visually distinct forms; the latter are encoded separately. See *Table 17-6*.

Table 17-6. Kawi Independent Vowels with Composite Representations

Vowel	Visually Distinct	Composite
aa	अ 11F05	अा <11F04, 11F34>
ii	꠨ 11F07	꠨ा <11F06, 11F34>
uu	꠩ 11F09	꠩ा <11F08, 11F34>
au		꠨ा <11F10, 11F34>
eu		अ̊ <11F04, 11F40>
euu		अ̊ा <11F04, 11F40, 11F34>

Two vocalic liquid letters have conjunct forms—the nature of ◌꠩ is not entirely clear. They are shown in *Table 17-7*.

Table 17-7. Kawi Vocalic Liquids with Conjunct Forms

Vocalic Liquid	Base Form	Conjunct Form
Vocalic *r*	ꠍ	◌ꠍ
Vocalic *l*	ꠎ	◌ꠎ

Dependent Vowels. The dependent vowels *o*, *au*, and *euu* are visually composites of other letters and dependent vowels, and are encoded as such, as shown in *Table 17-8*.

Table 17-8. Kawi Dependent Vowels with Composite Representations

Vowel	Composite
o	ꠦ◌ा <11F3E, 11F34>
au	ꠦ◌ा <11F3F, 11F34>
euu	◌̊ा <11F40, 11F34>

The dependent vowel *aa* has several glyph variants. The primary form is U+11F34 ○ꜟ KAWI VOWEL SIGN AA; one alternate form has been encoded as U+11F35 ○ꜟ KAWI VOWEL SIGN ALTERNATE AA, as its use may be required to avoid confusability. Other variants, such as ○, may be supported as stylistic variants.

The dependent vowel ○ꜟ *aa* has been repurposed as a consonant reduplicator in some manuscripts, and can in this case be combined with other vowels, for example, ꤷ *ppi* <11F26, 11F36, 11F34>.

The dependent vowels ○̊ *i* and ꤦ *u* are sometimes used together in a single cluster to mark the cluster as canceled and not meant to be read, for example, ꤴ for a misspelled *la*, *li*, or *lu*.

In some inscriptions, the dependent vowels ꤰ○ *ai* and ꤰ○ꜟ *au* are written with pre-base components that look similar to sequences of two dependent vowels ꤰ○ *e*. To transcribe these, use ꤰ○ twice: ꤰꤰ○ for ꤰ○ and ꤰꤰ○ꜟ for ꤰ○ꜟ.

Other Signs. U+11F00 ○̊ KAWI SIGN CANDRABINDU indicates nasalization in specific words such as ꤲꜟ *om*. U+11F01 ○̇ KAWI SIGN ANUSVARA represents final *-ŋ*, while U+11F03 ○: KAWI SIGN VISARGA represents final *-h*. U+11F5A ○̣ KAWI SIGN NUKTA is used to modify a few consonants to represent foreign sounds, typically coming from Arabic. For example, combining it with U+11F26 ꤦ KAWI LETTER PA results in ꤦ̣ *fa*.

Digits. The Kawi script has its own set of decimal digits. The digit U+11F52 ꤩ KAWI DIGIT TWO is used for the syllable *ro* in some manuscripts; additional marks can be attached to it in this usage.

Punctuation. Kawi materials use several punctuation characters to divide text into sections.

U+11F48 ꤳ KAWI PUNCTUATION SPACE FILLER is used to justify texts or fill gaps that are too small to fit another letter in the middle or at the end of a line. This character looks like U+11F54 ꤳ KAWI DIGIT FOUR in some inscriptions, but differs in others.

U+11F45 ꤵ KAWI PUNCTUATION SECTION MARKER, U+11F46 ꤵ KAWI PUNCTUATION ALTERNATE SECTION MARKER (which differs from U+11F45 in having some additional flourish), U+11F4E ⦿ KAWI PUNCTUATION SPIRAL, and U+11F4F ⦿~ KAWI PUNCTUATION CLOSING SPIRAL are similar in function to *siddham* signs in various other scripts, which are generally used as invocations at the beginning of texts. The Kawi analogues to the *siddham* signs have several distinct variants, which are often used in combination with other punctuation marks to indicate opening, closing, and major breaks in a text, such as ꤵ:ꤵ or ⊛⊙⊛.

Encoding Order and Rendering. Information on the encoding order of syllable components and on rendering is available in Unicode Technical Note #48, "Implementing Kawi."

Line Breaking. Opportunities for line breaking occur after any full orthographic syllable. Hyphens are not used.

Chapter 18

East Asia

This chapter presents scripts used in East Asia. This includes major writing systems associated with Chinese, Japanese, and Korean. It also includes several scripts for minority languages spoken in southern China, as well as the historic Khitan Small Script of northern China, and the historic Tangut script.

The characters that are now called East Asian ideographs, and known as Han ideographs in the Unicode Standard, were developed in China in the second millennium BCE. The basic system of writing Chinese using ideographs has not changed since that time, although the set of ideographs used, their specific shapes, and the technologies involved have developed over the centuries. The encoding of Chinese ideographs in the Unicode Standard is described in *Section 18.1, Han*. For more on usage of the term *ideograph*, see "Logosyllabaries" in *Section 6.1, Writing Systems*.

As civilizations developed surrounding China, they frequently adapted China's ideographs for writing their own languages. Japan, Korea, and Vietnam all borrowed and modified Chinese ideographs for their own languages. Chinese is an isolating language, monosyllabic and noninflecting, and ideographic writing suits it well. As Han ideographs were adopted for unrelated languages, however, extensive modifications were required.

Chinese ideographs were originally used to write Japanese, for which they are, in fact, ill suited. As an adaptation, the Japanese developed two syllabaries, *Hiragana* and *Katakana*, whose shapes are simplified or stylized versions of certain ideographs. (See *Section 18.4, Hiragana and Katakana*.) Chinese ideographs are called *kanji* in Japanese and are still used, in combination with *Hiragana* and *Katakana*, in modern Japanese.

In Korea, Chinese ideographs were originally used to write Korean, for which they are also ill suited. The Koreans developed an alphabetic system, *Hangul*, discussed in *Section 18.6, Hangul*. The shapes of Hangul syllables or the letter-like *jamos* from which they are composed are not directly influenced by Chinese ideographs. However, the individual jamos are grouped into syllabic blocks that resemble ideographs both visually and in the relationship they have to the spoken language (one syllable per block). Chinese ideographs are called *hanja* in Korean and are still used together with Hangul in South Korea for modern Korean. The Unicode Standard includes a complete set of Korean Hangul syllables as well as the individual jamos,

which can also be used to write Korean. *Section 3.12, Conjoining Jamo Behavior*, describes how to use the conjoining jamos and how to convert between the two methods for representing Korean.

In Vietnam, a set of native ideographs was created for Vietnamese based on the same principles used to create new ideographs for Chinese. These Vietnamese ideographs were used through the beginning of the 20th century and are occasionally used in more recent signage and other limited contexts.

Yi was originally written using a set of ideographs invented in imitation of the Chinese. Modern Yi as encoded in the Unicode Standard is a syllabary derived from these ideographs and is discussed in *Section 18.7, Yi*.

Bopomofo, discussed in *Section 18.3, Bopomofo*, is another recently invented syllabic system, used to represent Chinese phonetics.

In all these East Asian scripts, the characters (Chinese ideographs, Japanese *kana*, Korean Hangul syllables, and Yi syllables) are written within uniformly sized rectangles, usually squares. Traditionally, the basic writing direction followed the conventions of Chinese handwriting, in top-down vertical lines arranged from right to left across the page. Under the influence of Western printing technologies, a horizontal, left-to-right directionality has become common, and proportional fonts are seeing increased use, particularly in Japan. Horizontal, right-to-left text is also found on occasion, usually for shorter texts such as inscriptions or store signs. Diacritical marks are rarely used, although phonetic annotations are not uncommon. Older editions of the Chinese classics sometimes use the ideographic tone marks (U+302A..U+302D) to indicate unusual pronunciations of characters.

Many older character sets include characters intended to simplify the implementation of East Asian scripts, such as variant punctuation forms for text written vertically, halfwidth forms (which occupy only half a rectangle), and fullwidth forms (which allow Latin letters to occupy a full rectangle). These characters are included in the Unicode Standard for compatibility with older standards.

Appendix E, Han Unification History, describes how the diverse typographic traditions of mainland China, Taiwan, Japan, Korea, and Vietnam have been reconciled to provide a common set of ideographs in the Unicode Standard for all these languages and regions.

Nüshu is a siniform script devised by and for women to write the local Chinese dialect of southeastern Hunan province, China. Nüshu is based on Chinese Han characters. Unlike Chinese, the characters typically denote the phonetic value of syllables. Less often Nüshu characters are used as ideographs. Although very few fluent Nüshu users were alive in the late twentieth century, the script has drawn national and international attention, leading to the study and preservation of the script.

The Lisu script was developed in the early 20th century by using a combination of Latin letters, rotated Latin letters, and Latin punctuation repurposed as tone letters, to create a writing system for the Lisu language, spoken by large communities, mostly in Yunnan province in China. It sees considerable use in China, where it has been an official script since 1992.

The Miao script was created in 1904 by adapting Latin letter variants, English short-hand characters, Miao pictographs, and Cree syllable forms. The script was originally developed to write the Northeast Yunnan Miao language of southern China. Today it is also used to write other Miao dialects and the languages of the Yi and Lisu nationalities of southern China.

Tangut is a large, historic siniform ideographic script used to write the Tangut language, a Tibeto-Burman language spoken from about the 11th century CE until the 16th century in the area of present-day northwestern China. Tangut was re-discovered in the late 19th century, and has been largely deciphered. Today the script is of interest to students and scholars.

Khitan Small Script was created about 925 CE, and was one of two scripts used by the Khitan people of Northern China to write the Khitan language during the Liao dynasty, the Qara Khitai empire, and the Jin dynasty. It is only partially deciphered. The script contains logograms and phonograms written in vertical columns, running right to left, similar to how Chinese is traditionally written.

18.1 Han

18.1.1 CJK Unified Ideographs

The Unicode Standard contains a set of unified Han ideographic characters used in the written Chinese, Japanese, and Korean languages. The term *Han*, derived from the Chinese Han Dynasty, refers generally to Chinese traditional culture. The Han ideographic characters make up a coherent script, which was traditionally written vertically, with the vertical lines ordered from right to left. In modern usage, especially in technical works and in computer-rendered text, the Han script is written horizontally from left to right and is freely mixed with Latin or other scripts. When used in writing Japanese or Korean, the Han characters are interspersed with other scripts unique to those languages (Hiragana and Katakana for Japanese; Hangul syllables for Korean).

Although the term "CJK"—Chinese, Japanese, and Korean—is used throughout this text to describe the languages that currently use Han ideographic characters, it should be noted that earlier Vietnamese writing systems were based on Han ideographs. Consequently, the term "CJKV" would be more accurate in a historical sense. Han ideographs are still used for historical, religious, and pedagogical purposes in Vietnam. For more on usage of the term *ideograph*, see "Logosyllabaries" in *Section 6.1, Writing Systems*.

The term "Han ideographic characters" is used within the Unicode Standard as a common term traditionally used in Western texts, although "sinogram" is preferred by professional linguists. Taken literally, the word "ideograph" applies only to some of the ancient original character forms, which indeed arose as ideographic depictions. The vast majority of Han characters were developed later via composition, borrowing, and other non-ideographic principles, but the term "Han ideographs" remains in English usage as a conventional cover term for the script as a whole.

The Han ideographic characters constitute a very large set, numbering in the tens of thousands. They have a long history of use in East Asia. Enormous compendia of Han ideographic characters exist because of a continuous, millennia-long scholarly tradition of collecting all Han character citations, including variant, mistaken, and nonce forms, into annotated character dictionaries.

The Unicode Standard draws its unified Han character repertoire from a number of different character set standards. These standards are grouped into a number of sources listed in tables in *Appendix E.3, CJK Sources*.

Because of the large size of the Han ideographic character repertoire, and because of the particular problems that the characters pose for standardizing their encoding, this character block description is more extended than that for other scripts and is

divided into several subsections. The first subsection, "Blocks Containing Han Ideographs," describes the way in which the Unicode Standard divides Han ideographs into blocks. This subsection is followed by an extended discussion of the characteristics of Han characters, with particular attention being paid to the problem of unification of encoding for characters used for different languages. There is a formal statement of the principles behind the Unified Han character encoding adopted in the Unicode Standard and the order of its arrangement. For a detailed account of the background and history of development of the Unified Han character encoding, see *Appendix E, Han Unification History*.

18.1.2 Blocks Containing Han Ideographs

Han ideographic characters are found in several blocks of the Unicode Standard, as shown in *Table 18-1*.

Table 18-1. Blocks Containing Han Ideographs

Block	Range	Comment
CJK Unified Ideographs	4E00–9FFF	Common
CJK Unified Ideographs Extension A	3400–4DBF	Rare
CJK Unified Ideographs Extension B	20000–2A6DF	Rare, historic
CJK Unified Ideographs Extension C	2A700–2B73F	Rare, historic
CJK Unified Ideographs Extension D	2B740–2B81F	Urgently needed
CJK Unified Ideographs Extension E	2B820–2CEAF	Rare, historic
CJK Unified Ideographs Extension F	2CEB0–2EBEF	Rare, historic
CJK Unified Ideographs Extension G	30000–3134F	Rare, historic
CJK Unified Ideographs Extension H	31350–323AF	Rare, historic
CJK Unified Ideographs Extension I	2EBF0–2EE5F	Urgently needed
CJK Compatibility Ideographs	F900–FAFF	Duplicates, unifiable variants, corporate characters
CJK Compatibility Ideographs Supplement	2F800–2FA1F	Unifiable variants

Characters in the unified ideograph blocks are defined by the IRG, based on Han unification principles explained later in this section.

The two compatibility ideographs blocks contain various duplicate or unifiable variant characters encoded for round-trip compatibility with various legacy standards. For historic reasons, the CJK Compatibility Ideographs block also contains twelve CJK unified ideographs. Those twelve ideographs are clearly labeled in the code charts for that block.

Extensions to the URO. The initial repertoire of the CJK Unified Ideographs block included characters submitted to the IRG prior to 1992, consisting of commonly used characters. That initial repertoire, also known as the Unified Repertoire and

Ordering, or URO, was derived entirely from the G, T, J, and K sources. The repertoire in the CJK Unified Ideographs block has subsequently been extended with small sets of unified ideographs or ideographic components needed for interoperability with various standards, or for other reasons, as shown in *Table 18-2*. The range U+9FFD..U+9FFF filled the reserved space at the end of this block.

Table 18-2. Small Extensions to CJK Blocks

Range	Version	Comment
9FA6–9FB3	4.1	Interoperability with HKSCS standard
9FB4–9FBB	4.1	Interoperability with GB 18030 standard
9FBC–9FC2	5.1	Interoperability with commercial implementations
9FC3	5.1	Correction of mistaken unification
9FC4–9FC6	5.2	Interoperability with ARIB standard
9FC7–9FCB	5.2	Interoperability with HKSCS standard
9FCC	6.1	Interoperability with commercial implementations
9FCD–9FCF	8.0	Interoperability with TGH 2013 standard
9FD0	8.0	Correction of mistaken unification
9FD1–9FD5	8.0	Miscellaneous urgently needed characters
9FD6–9FE9	10.0	Ideographs for Slavonic transcription
9FEA	10.0	Correction of mistaken unification
9FEB–9FED	11.0	Ideographs for chemical elements
9FEE–9FEF	11.0	Interoperability with government implementations
9FF0–9FFC	13.0	Zoological, chemical, and geological terms
9FFD–9FFF	14.0	Interoperability with government implementations
4DB6–4DBF	13.0	Corrections of mistaken unifications
2A6D7–2A6DD	13.0	Gongche characters for Kunqu Opera
2A6DE–2A6DF	14.0	Interoperability with government implementations
2B735–2B736	14.0	Corrections of mistaken unifications
2B737	14.0	Urgently needed character
2B738	14.0	Correction of mistaken unification
2B739	15.0	Urgently needed character

Extensions to Other CJK Blocks. Starting with Version 13.0, some of the small repertoire extensions have involved reserved ranges at the end of other CJK blocks. Those ranges are also shown in *Table 18-2*. The range U+4DB6..U+4DBF filled the reserved space at the end of the CJK Unified Ideographs Extension A block, the range U+2A6DE..U+2A6DF filled the reserved space at the end of the CJK Unified Ideographs Extension B block, and the range U+2B735..U+2B739 used reserved space at the end of the CJK Unified Ideographs Extension C block.

Han Ideographs for Slavonic Transcription. The URO includes twenty CJK Unified Ideographs, U+9FD6 through U+9FE9, which are used for transcribing Slavonic literary documents into Chinese. Renewed contact between the Russian and Chinese Empires from the 18th to the 20th centuries led to the translation of Slavonic literary documents into both classical and vernacular Chinese. The Russian Mission in Beijing was a driving force behind this effort, and many of these characters were coined by Archimandrite Gurias, who was the head of the 14th Russian Mission (1858–1864). Although some existing CJK Unified Ideographs can be used for transcribing Slavonic, these twenty characters are distinct. Many of these characters are unusual in that they represent syllables not usually found in Chinese.

Other Large CJK Extensions. Characters in the CJK Unified Ideographs Extension A block are rare and are not unifiable with characters in the CJK Unified Ideographs block. They were submitted to the IRG during 1992–1998 and are derived entirely from the G, T, J, K, and V sources.

The CJK Unified Ideographs Extension B block contains rare and historic characters that are also not unifiable with characters in the CJK Unified Ideographs block. They were derived from versions of national standards submitted to the IRG during 1998–2000. The characters encoded in Extension B may, in some instances, differ slightly from published versions of those standards.

The CJK Unified Ideographs Extension C through I blocks mostly contain rare, historic, uncommon, or urgently needed characters that are not unifiable with characters in any previously encoded CJK Unified Ideographs block. Extension D and Extension I are somewhat unique in that they are made up of urgently needed characters from various regions. Extension C ideographs were submitted to the IRG during 2002–2006. Extension D ideographs were submitted to the IRG during 2006–2009. Extension E ideographs were submitted to the IRG during 2006–2013. Extension F ideographs were submitted during 2012–2015. Extension G ideographs were submitted during 2015. Extension H ideographs were submitted during 2017. Extension I is unique, in that it consists entirely of urgently needed characters from China.

Principles for Extensions. The only principled difference in the unification work done by the IRG on the unified ideograph blocks is that the Source Separation Rule (rule R1) was applied only to the original CJK Unified Ideographs block and not to the extension blocks. The Source Separation Rule states that ideographs that are distinctly encoded in a source must not be unified. (For further discussion, see "Principles of Han Unification" later in this section.)

The unified ideograph blocks are not closed repertoires. Each may contain a small range of reserved code points at the end of the block. Additional unified ideographs may eventually be encoded in those ranges—as has already occurred in the CJK Unified Ideographs block, as well as in Extensions A through C. There is no guarantee that any such Han ideographic additions would be of the same types or from

the same sources as preexisting characters in the block, and implementations should be careful not to make hard-coded assumptions regarding the range of assignments within the Han ideographic blocks in general.

Several Han characters unique to the U source and which are not unifiable with other characters in the CJK Unified Ideographs block are found in the CJK Compatibility Ideographs block. There are 12 of these characters: U+FA0E, U+FA0F, U+FA11, U+FA13, U+FA14, U+FA1F, U+FA21, U+FA23, U+FA24, U+FA27, U+FA28, and U+FA29. The remaining characters in the CJK Compatibility Ideographs block and the CJK Compatibility Ideographs Supplement block are either duplicates or unifiable variants of a character in one of the blocks of unified ideographs.

IICore. IICore (International Ideograph Core) is a set of important Han ideographs, incorporating characters from all the defined blocks. This set of nearly 10,000 characters has been developed by the IRG and represents the set of characters in everyday use throughout East Asia. By covering the characters in IICore, developers guarantee that they can handle all the needs of almost all of their customers. This coverage is of particular use on devices such as cell phones or PDAs, which have relatively stringent resource limitations. Characters in IICore are explicitly tagged as such in the Unihan Database (see Unicode Standard Annex #38, "Unicode Han Database (Unihan)").

UnihanCore2020. UnihanCore2020 is a set of over 20,000 Han ideographs. The set includes 68 compatibility characters necessary for some regions. Like IICore, this set is intended to cover the needs of customers in East Asia, but its repertoire is much larger because of the increased memory and storage capacity of contemporary hardware, including mobile devices. The repertoire of the UnihanCore2020 subset is identified with the kUnihanCore2020 key in the Unihan Database. See Unicode Standard Annex #38, "Unicode Han Database (Unihan)".

18.1.3 General Characteristics of Han Ideographs

The authoritative Japanese dictionary *Koujien* (1983) defines Han characters to be:

> ...characters that originated among the Chinese to write the Chinese language. They are now used in China, Japan, and Korea. They are logographic (each character represents a word, not just a sound) characters that developed from pictographic and ideographic principles. They are also used phonetically. In Japan they are generally called *kanji* (Han, that is, Chinese, characters) including the "national characters" (*kokuji*) such as *touge* (mountain pass), which have been created using the same principles.

For many centuries, written Chinese was the accepted written standard throughout East Asia. The influence of the Chinese language and its written form on the modern

East Asian languages is similar to the influence of Latin on the vocabulary and written forms of languages in the West. This influence is immediately visible in the mixture of Han characters and native phonetic scripts (*kana* in Japan, *hangul* in Korea) as now used in the orthographies of Japan and Korea (see *Table 18-3*).

Table 18-3. Common Han Characters

Han Character	Chinese	Japanese	Korean	English Translation
天	tiān	ten, ame	chen	heaven, sky
地	dì	chi, tsuchi	ji	earth, ground
人	rén	jin, hito	in	man, person
山	shān	san, yama	san	mountain
水	shuǐ	sui, mizu	su	water
上	shàng	jou, ue	sang	above
下	xià	ka, shita	ha	below

The evolution of character shapes and semantic drift over the centuries has resulted in changes to the original forms and meanings. For example, the Chinese character 湯 *tāng* (Japanese *tou* or *yu*, Korean *thang*), which originally meant "hot water," has come to mean "soup" in Chinese. "Hot water" remains the primary meaning in Japanese and Korean, whereas "soup" appears in more recent borrowings from Chinese, such as "soup noodles" (Japanese *tanmen*; Korean *thangmyen*). Still, the identical appearance and similarities in meaning are dramatic and more than justify the concept of a unified Han script that transcends language.

The "nationality" of the Han characters became an issue only when each country began to create coded character sets (for example, China's GB 2312-80, Japan's JIS X 0208-1978, and Korea's KS C 5601-87) based on purely local needs. This problem appears to have arisen more from the priority placed on local requirements and lack of coordination with other countries, rather than out of conscious design. Nevertheless, the identity of the Han characters is fundamentally independent of language, as shown by dictionary definitions, vocabulary lists, and encoding standards.

Terminology. Several standard romanizations of the term used to refer to East Asian ideographic characters are commonly used. They include *hànzì* (Chinese), *kanzi* (Japanese), *kanji* (colloquial Japanese), *hanja* (Korean), and *Chữ hán* (Vietnamese). The standard English translations for these terms are interchangeable: Han character, Han ideographic character, East Asian ideographic character, or CJK ideographic character. For clarity, the Unicode Standard uses some subset of the English terms when referring to these characters. The term *Kanzi* is used in reference to a specific Japanese government publication. The unrelated term *Kangxi* (which is a Chinese reign name, rather than another romanization of "Han character") is used only when referring to the primary dictionary used for determining Han character arrangement in the Unicode Standard. (See *Table 18-7*.)

Distinguishing Han Character Usage Between Languages. There is some concern that unifying the Han characters may lead to confusion because they are sometimes used differently by the various East Asian languages. Computationally, Han character unification presents no more difficulty than employing a single Latin character set that is used to write languages as different as English and French. Programmers do not expect the characters "c", "h", "a", and "t" alone to tell us whether *chat* is a French word for cat or an English word meaning "informal talk." Likewise, we depend on context to identify the American hood (of a car) with the British bonnet. Few computer users are confused by the fact that ASCII can also be used to represent such words as the Welsh word *ynghyd*, which are strange looking to English eyes. Although it would be convenient to identify words by language for programs such as spell-checkers, it is neither practical nor productive to encode a separate Latin character set for every language that uses it.

Similarly, the Han characters are often combined to "spell" words whose meaning may not be evident from the constituent characters. For example, the two characters "to cut" and "hand" mean "postage stamp" in Japanese, but the compound may appear to be nonsense to a speaker of Chinese or Korean (see *Figure 18-1*).

Figure 18-1. Han Spelling

切 + 手 = 1. Japanese "stamp"
to cut   hand   2. Chinese "cut hand"

Even within one language, a computer requires context to distinguish the meanings of words represented by coded characters. The word *chuugoku* in Japanese, for example, may refer to China or to a district in central west Honshuu (see *Figure 18-2*).

Figure 18-2. Semantic Context for Han Characters

中 + 国 = 1. China
middle   country   2. Chuugoku district of Honshuu

Coding these two characters as four so as to capture this distinction would probably cause more confusion and still not provide a general solution. The Unicode Standard leaves the issues of language tagging and word recognition up to a higher level of software and does not attempt to encode the language of the Han characters.

Simplified and Traditional Chinese. There are currently two main varieties of written Chinese: "simplified Chinese" (*jiǎntǐzì*), used in most parts of the People's Republic of China (PRC) and Singapore, and "traditional Chinese" (*fántǐzì*), used predominantly in the Hong Kong and Macao SARs, Taiwan, and overseas Chinese communities. The process of interconverting between the two is a complex one. This complexity arises largely because a single simplified form may correspond to

multiple traditional forms, such as U+53F0 台, which is a traditional character in its own right and the simplified form for U+6AAF 檯, U+81FA 臺, and U+98B1 颱. Moreover, vocabulary differences have arisen between Mandarin as spoken in Taiwan and Mandarin as spoken in the PRC, the most notable of which is the usual name of the language itself: *guóyǔ* (the National Language) in Taiwan and *pǔtōnghuà* (the Common Speech) in the PRC. Merely converting the character content of a text from simplified Chinese to the appropriate traditional counterpart is insufficient to change a simplified Chinese document to traditional Chinese, or vice versa. (The vast majority of Chinese characters are the same in both simplified and traditional Chinese.)

There are two PRC national standards, GB 2312-80 and GB 12345-90, which are intended to represent simplified and traditional Chinese, respectively. The character repertoires of the two are the same, but the simplified forms occur in GB 2312-80 and the traditional ones in GB 12345-90. These are both part of the IRG G source, with traditional forms and simplified forms separated where they differ. As a result, the Unicode Standard contains a number of distinct simplifications for characters, such as U+8AAC 說 and U+8BF4 说.

While there are lists of official simplifications published by the PRC, most of these are obtained by applying a few general principles to specific areas. In particular, there is a set of radicals (such as U+2F94 言 KANGXI RADICAL SPEECH, U+2F99 貝 KANGXI RADICAL SHELL, U+2FA8 門 KANGXI RADICAL GATE, and U+2FC3 鳥 KANGXI RADICAL BIRD) for which simplifications exist (U+2EC8 讠 CJK RADICAL C-SIMPLIFIED SPEECH, U+2EC9 贝 CJK RADICAL C-SIMPLIFIED SHELL, U+2ED4 门 CJK RADICAL C-SIMPLIFIED GATE, and U+2EE6 鸟 CJK RADICAL C-SIMPLIFIED BIRD). The basic technique for simplifying a character containing one of these radicals is to substitute the simplified radical, as in the previous example.

The Unicode Standard does not explicitly encode all simplified forms for traditional Chinese characters. Where the simplified and traditional forms exist as different encoded characters, each should be used as appropriate. The Unicode Standard does not specify how to represent a new simplified form (or, more rarely, a new traditional form) that can be derived algorithmically from an encoded traditional form (simplified form).

Early Forms of Chinese. Prior to the 20th century, the standard form of written Chinese was literary Chinese, a form derived from the classical Chinese that was written, but probably not spoken, by Confucius in the sixth century BCE.

The repertoire of CJK unified ideographs encoded in the Unicode Standard covers modern Chinese, literary Chinese, and classical Chinese.

Sorting Han Ideographs. The Unicode Standard does not define a method by which ideographic characters are sorted; the requirements for sorting differ by locale and

application. Possible collating sequences include phonetic, radical-stroke (*Kangxi*, *Xinhua Zidian*, and so on), four-corner, and total stroke count. Raw character codes alone are seldom sufficient to achieve a usable ordering in any of these schemes; ancillary data are usually required. (See *Table 18-7* for a summary of the authoritative sources used to determine the order of Han ideographs in the code charts.)

Character Glyphs. In form, Han characters are monospaced. Every character takes the same vertical and horizontal space, regardless of how simple or complex its particular form is. This practice follows from the long history of printing and typographical practice in China, which traditionally placed each character in a square cell. When written vertically, there are also a number of named cursive styles for Han characters, but the cursive forms of the characters tend to be quite idiosyncratic and are not implemented in general-purpose Han character fonts for computers.

There may be a wide variation in the glyphs used in different countries and for different applications. The most commonly used typefaces in one country may not be used in others.

The types of glyphs used to depict characters in the Han ideographic repertoire of the Unicode Standard have been constrained by available fonts. Users are advised to consult authoritative sources for the appropriate glyphs for individual markets and applications. It is assumed that most Unicode implementations will provide users with the ability to select the font (or mixture of fonts) that is most appropriate for a given locale.

18.1.4 Principles of Han Unification

Three-Dimensional Conceptual Model. To develop the explicit rules for unification, a conceptual framework was developed to model the nature of Han ideographic characters. This model expresses written elements in terms of three primary attributes: semantic (meaning, function), abstract shape (general form), and actual shape (instantiated, typeface form). These attributes are graphically represented in three dimensions according to the *X*, *Y*, and *Z* axes (see *Figure 18-3*).

Figure 18-3. Three-Dimensional Conceptual Model
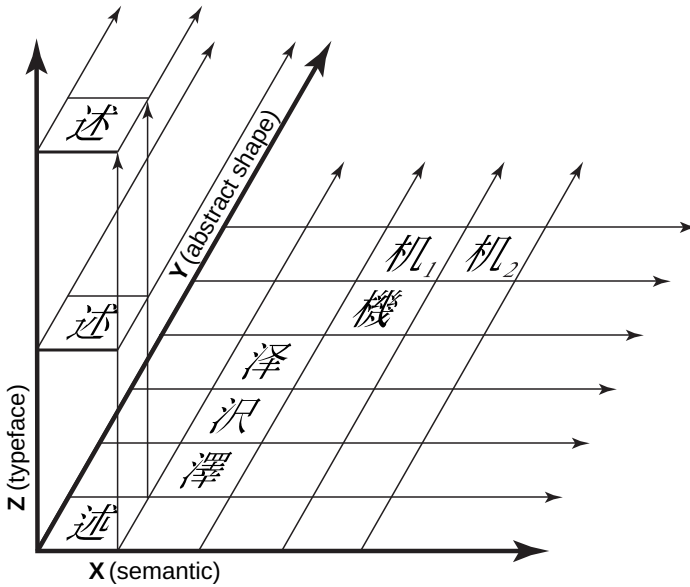
The semantic attribute (represented along the *X* axis) distinguishes characters by meaning and usage. Distinctions are made between entirely unrelated characters such as 澤 (marsh) and 機 (machine) as well as extensions or borrowings beyond the original semantic cluster such as 机$_1$ (a phonetic borrowing used as a simplified form of 機) and 机$_2$ (table, the original meaning).

The abstract shape attribute (the *Y* axis) distinguishes the variant forms of a single character with a single semantic attribute (that is, a character with a single position on the *X* axis).

The actual shape (typeface) attribute (the *Z* axis) is for differences of type design (the actual shape used in imaging) of each variant form.

Z-axis typeface and stylistic differences are generally ignored for the purpose of encoding Han ideographs, but can be represented in text by the use of variation sequences; see *Section 23.4, Variation Selectors*.

18.1.5 Unification Rules

The following rules were applied during the process of merging Han characters from the different source character sets.

> **R1 Source Separation Rule. If two ideographs are distinct in a primary source standard, then they are not unified.**

- This rule is sometimes called the *round-trip rule* because its goal is to facilitate a round-trip conversion of character data between an IRG source standard and the Unicode Standard without loss of information.

- This rule was applied only for the work on the original CJK Unified Ideographs block [also known as the Unified Repertoire and Ordering (URO)]. The IRG dropped this rule in 1992 and will not use it in future work.

Figure 18-4 illustrates six variants of the CJK ideograph meaning "sword."

Figure 18-4. CJK Source Separation

剣　劍　剏　劔　劒　釼
5263　528D　5271　5294　5292　91FC
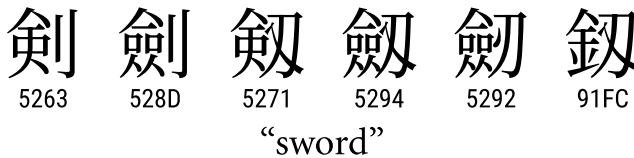
"sword"

Each of the six variants in *Figure 18-4* is separately encoded in one of the primary source standards—in this case, J0 (JIS X 0208-1990), as shown in *Table 18-4*.

Table 18-4. Source Encoding for Sword Variants

Unicode	JIS
U+5263	J0-3775
U+528D	J0-5178
U+5271	J0-517B
U+5294	J0-5179
U+5292	J0-517A
U+91FC	J0-6E5F

Because the six sword characters are historically related, they are not subject to disunification by the Noncognate Rule (R2) and thus would ordinarily have been considered for possible abstract shape-based unification by R3. Under that rule, the fourth and fifth variants would probably have been unified for encoding. However, the Source Separation Rule required that all six variants be separately encoded, precluding them from any consideration of shape-based unification. Further variants of the "sword" ideograph, U+5251 and U+528E, are also separately encoded because of application of the Source Separation Rule—in that case applied to one or more Chinese primary source standards, rather than to the J0 Japanese primary source standard.

R2 **Noncognate Rule. In general, if two ideographs are unrelated in historical derivation (noncognate characters), then they are not unified.**

For example, the ideographs in *Figure 18-5*, although visually quite similar, are nevertheless not unified because they are historically unrelated and have distinct meanings.

Figure 18-5. Not Cognates, Not Unified

土 ≠ 士
earth   warrior, scholar
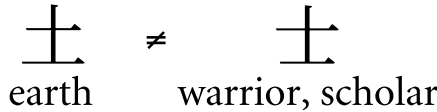
R3 **By means of a two-level classification (described next), the abstract shape of each ideograph is determined. Any two ideographs that possess the same abstract shape are then unified provided that their unification is not disallowed by either the Source Separation Rule or the Noncognate Rule.**

18.1.6 Abstract Shape

Two-Level Classification. Using the three-dimensional model, characters are analyzed in a two-level classification. The two-level classification distinguishes characters by abstract shape (*Y* axis) and actual shape of a particular typeface (*Z* axis). Variant forms are identified based on the difference of abstract shapes.

To determine differences in abstract shape and actual shape, the structure and features of each component of an ideograph are analyzed as follows.

Ideographic Component Structure. The component structure of each ideograph is examined. A component is a geometrical combination of primitive elements. Various ideographs can be configured with these components used in conjunction with other components. Some components can be combined to make a component more complicated in its structure. Therefore, an ideograph can be defined as a component tree with the entire ideograph as the root node and with the bottom nodes consisting of primitive elements (see *Figure 18-6* and *Figure 18-7*).

Figure 18-6. Ideographic Component Structure
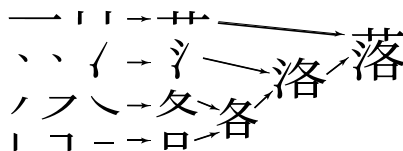
Figure 18-7. The Most Superior Node of an Ideographic Component

棺 vs. 棺 ⟨ 木
　　　　　皂 vs. 皀 ⟨ 白
　　　　　　　　　七 vs. 匕
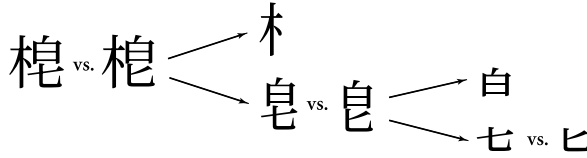
Ideograph Features. The following features of each ideograph to be compared are examined:

- Number of components
- Relative positions of components in each complete ideograph
- Structure of a corresponding component
- Treatment in a source character set
- Radical contained in a component

Uniqueness or Unification. If one or more of these features are different between the ideographs compared, the ideographs are considered to have different abstract shapes and, therefore, are considered unique characters and are not unified. If all of these features are identical between the ideographs, the ideographs are considered to have the same abstract shape and are unified.

Spatial Positioning. Ideographs may exist as a unit or may be a component of more complex ideographs. A source standard may describe a requirement for a component with a specific spatial positioning that would be otherwise unified on the principle of having the same abstract shape as an existing full ideograph. Examples of spatial positioning for ideographic components are left half, top half, and so on.

Examples. The examples in *Table 18-5* illustrate the reasons for not unifying characters, including typical differences in abstract character shape.

Table 18-5. Ideographs Not Unified

Characters	Reason
日≠日	Non-cognate characters
說≠説	Characters treated as distinct in a source character set
崖≠厓	Different number of components
峰≠峯	Same number of components placed in different relative positions
拡≠擴	Same number and same relative position of components, corresponding components structured differently
祕≠秘	Characters with different radical in a component

Differences in the actual shapes of ideographs that *have* been unified are illustrated in *Table 18-6*.

Table 18-6. Ideographs Unified

Characters	Reason
周≈周	Different writing sequence
雪≈雪	Differences in overshoot at the stroke termination
西≈酉	Differences in contact of strokes
巨≈巨	Differences in protrusion at the folded corner of strokes
堊≈堊	Differences in bent strokes
朱≈朱	Differences in stroke termination
父≈父	Differences in accent at the stroke initiation
八≈八	Difference in rooftop modification
說≈説	Difference in rotated strokes/dots[†]

[†] These ideographs (having the same abstract shape) would have been unified except for the Source Separation Rule.

18.1.7 Han Ideograph Arrangement

The arrangement of the Unicode Han characters is based on the positions of characters as they are listed in four major dictionaries. The *Kangxi Zidian* was chosen as primary because it contains most of the source characters and because the dictionary itself and the principles of character ordering it employs are commonly used throughout East Asia.

The Han ideograph arrangement follows the index (page and position) of the dictionaries listed in *Table 18-7* with their priorities.

Table 18-7. Han Ideograph Arrangement

Priority	Dictionary	City	Publisher	Version
1	*Kangxi Zidian*	Beijing	Zhonghua Bookstore, 1989	Seventh edition
2	*Dai Kan-Wa Jiten*	Tokyo	Taishuukan Shoten, 1986	Revised edition
3	*Hanyu Da Zidian*	Chengdu	Sichuan Cishu Publishing, 1986	First edition
4	*Dae Jaweon*	Seoul	Samseong Publishing Co. Ltd, 1988	First edition

When a character is found in the *Kangxi Zidian*, it follows the *Kangxi Zidian* order. When it is not found in the *Kangxi Zidian* and it is found in *Dai Kan-Wa Jiten*, it is

given a position extrapolated from the *Kangxi* position of the preceding character in *Dai Kan-Wa Jiten*. When it is not found in either *Kangxi* or *Dai Kan-Wa*, then the *Hanyu Da Zidian* and *Dae Jaweon* dictionaries are consulted in a similar manner.

Ideographs with simplified *Kangxi* radicals are placed in a group following the traditional *Kangxi* radical from which the simplified radical is derived. For example, characters with the simplified radical 讠 corresponding to *Kangxi* radical 言 follow the last nonsimplified character having 言 as a radical. The arrangement for these simplified characters is that of the *Hanyu Da Zidian*.

The few characters that are not found in any of the four dictionaries are placed following characters with the same *Kangxi* radical and stroke count. The radical-stroke order that results is a culturally neutral order. It does not exactly match the order found in common dictionaries.

Information for sorting all CJK ideographs by the radical-stroke method is found in the Unihan Database (see Unicode Standard Annex #38, "Unicode Han Database (Unihan)"). It should be used if characters from the various blocks containing ideographs (see *Table 18-1*) are to be properly interleaved. Note, however, that there is no standard way of ordering characters with the same radical-stroke count; for most purposes, Unicode code point order would be as acceptable as any other way.

Details regarding the form of the online charts for the CJK unified ideographs are discussed in *Section 24.2, CJK and Other Ideographs*.

18.1.8 Radical-Stroke Indices

Various radical-stroke indices are provided on the Unicode website to ease the search for particular Han ideographs in the Unicode Standard. An interactive radical-stroke index page enables queries by specific Kangxi radical numbers and the number of residual strokes. Three radical-stroke indices are also provided in PDF format. The more extensive of them covers all of the ideographs in the CJK Unified Ideographs and CJK Compatibility Ideographs blocks. There are also more compact radical-stroke indices that are limited to the Han ideographs as specified by the IICore and UnihanCore2020 subsets.

The most authoritative source for radical-stroke information is the eighteenth-century *Kangxi* dictionary, which established the classification system of 214 radicals. The main issue with using Kangxi radicals today is that many simplified ideographs are difficult to classify under the system of 214 Kangxi radicals. As a result, various modern radical classification systems have been established. However, none of them is in general use, and the 214 Kangxi radicals remain the most universally recognized to this day. See "CJK and Kangxi Radicals" later in this section for more details.

According to the traditional radical-stroke classification system, each Han ideograph is considered to be written with a radical plus its residual strokes. For example, the ideograph 說 is assigned to the radical 言 and has seven residual strokes. To find the ideograph 說 in a dictionary, one would first locate the section for its radical, 言, and then find the subsection for ideographs with seven residual strokes. With the exception of ideographs that are classified under a simplified radical, simplified ideographs are generally classified under the same radical as their traditional forms. For example, the simplified ideograph 伲 and its traditional form, 儗, are *both* classified under the radical 人.

This classification system is complicated by the fact that there are occasional ambiguities in the counting of strokes of the radical itself or the ideograph's residual components. It is further complicated in that two or more ideograph dictionaries may disagree under which particular radical an ideograph is classified. Ideographs classified under more than one radical may thus appear more than once in the radical-stroke indices.

18.1.9 Mappings for Han Ideographs

The mappings defined by the IRG between the ideographs in the Unicode Standard and the IRG sources are specified in the Unihan Database. These mappings are considered to be normative parts of ISO/IEC 10646 and of the Unicode Standard; that is, the characters are *defined* to be the targets for conversion of these characters in these character set standards.

These mappings have been derived from editions of the source standards provided directly to the IRG by its member bodies, and they may not match mappings derived from the published editions of these standards. For this reason, developers may choose to use alternative mappings more directly correlated with published editions.

Specialized conversion systems may also choose more sophisticated mapping mechanisms—for example, semantic conversion, variant normalization, or conversion between simplified and traditional Chinese.

The Unicode Consortium also provides mapping information that extends beyond the normative mappings defined by the IRG. These additional mappings include mappings to character set standards included in the U source, including duplicate characters from KS C 5601-1987, mappings to portions of character set standards omitted from IRG sources, references to standard dictionaries, and suggested character/stroke counts.

18.1.10 CJK Compatibility Ideographs: U+F900–U+FAFF

The Korean national standard KS C 5601-1987 (now known as KS X 1001:1998), which served as one of the primary source sets for the Unified CJK Ideograph Repertoire

and Ordering, Version 2.0, contains 268 duplicate encodings of identical ideograph forms to denote alternative pronunciations. That is, in certain cases, the standard encodes a single character multiple times to denote different linguistic uses. This approach is like encoding the letter "a" five times to denote the different pronunciations it has in the words *hat*, *able*, *art*, *father*, and *adrift*. Because they are in all ways identical in shape to their nominal counterparts, they were excluded by the IRG from its sources. For round-trip conversion with KS C 5601-1987, they are encoded separately from the primary CJK Unified Ideographs block.

Another 34 ideographs from various regional and industry standards were encoded in this block, primarily to achieve round-trip conversion compatibility. Twelve of these ideographs (U+FA0E, U+FA0F, U+FA11, U+FA13, U+FA14, U+FA1F, U+FA21, U+FA23, U+FA24, U+FA27, U+FA28, and U+FA29) are not encoded in blocks for CJK unified ideographs. These 12 characters are not duplicates and should be treated as a small extension to the set of unified ideographs.

Except for the 12 unified ideographs just enumerated, CJK compatibility ideographs from this block are not used in Ideographic Description Sequences.

An additional 59 compatibility ideographs are found from U+FA30 to U+FA6A. They are included in the Unicode Standard to provide full round-trip compatibility with the ideographic repertoire of JIS X 0213:2000 and should not be used for any other purpose.

An additional three compatibility ideographs are encoded at the range U+FA6B to U+FA6D. They are included in the Unicode Standard to provide full round-trip compatibility with the ideographic repertoire of the Japanese television standard, ARIB STD-B24, and should not be used for any other purpose.

An additional 106 compatibility ideographs are encoded at the range U+FA70 to U+FAD9. They are included in the Unicode Standard to provide full round-trip compatibility with the ideographic repertoire of KPS 10721-2000. They should not be used for any other purpose.

The names for the compatibility ideographs are also algorithmically derived. Thus the name for the compatibility ideograph U+F900 is CJK COMPATIBILITY IDEO-GRAPH-F900. See the formal definition of the Name property in *Section 4.8, Name*.

All of the compatibility ideographs in this block, except for the 12 unified ideographs, have standardized variation sequences defined in StandardizedVariants.txt. See the discussion in *Section 23.4, Variation Selectors* for more details.

18.1.11 CJK Compatibility Supplement: U+2F800–U+2FA1D

The CJK Compatibility Ideographs Supplement block consists of additional compatibility ideographs required for round-trip compatibility with CNS 11643-1992, planes 3,

4, 5, 6, 7, and 15. They should not be used for any other purpose and, in particular, may not be used in Ideographic Description Sequences.

All of the additional compatibility ideographs in this block have standardized variation sequences defined in StandardizedVariants.txt. See the discussion in *Section 23.4, Variation Selectors* for more details.

18.1.12 Kanbun: U+3190–U+319F

This block contains a set of Kanbun marks that are used in Japanese literary texts to indicate the Japanese reading order of Classical Chinese poetry and prose. These marks, named for the Japanese word for Chinese writing (漢文), occur particularly in Japanese educational and scholastic texts. They are typically written in an annotation style, placed interlinearly at the left side of each line of vertically rendered original Chinese text. Typesetting Kanbun text is inherently complex, requiring some form of markup and special handling to achieve the desired layout results.

Fourteen of the Kanbun marks, in the range U+3192 ⁻ IDEOGRAPHIC ANNOTATION ONE MARK through U+319F ᐟ IDEOGRAPHIC ANNOTATION MAN MARK, have compatibility decompositions to a corresponding CJK unified ideograph. These marks are merely special-purpose variants of those CJK unified ideographs, used with a specialized meaning and layout rules in Kanbun text. The way the glyphs are shown in the code charts at reduced size and raised above the baseline is intended to mimic their appearance as formatted for use in annotations. This appearance is the reason the compatibility mappings have been assigned the tag <super>. The compatibility mappings do not imply that these characters are appropriate for use as superscript forms in ordinary Chinese text; the preferred means for that purpose are text styles or markup in rich text. (See *Section 22.4, Superscript and Subscript Symbols* for more information.) Common practice for existing Japanese fonts that support these characters is to provide their glyphs at full size, with the expectation that the layout engine will scale and position them accordingly, per the layout specification for Kanbun text in JIS X 4051.

18.1.13 Symbols Derived from Han Ideographs

A number of symbols derived from Han ideographs can be found in other blocks. See "Enclosed CJK Letters and Months: U+3200–U+32FF," "CJK Compatibility: U+3300–U+33FF," and "Enclosed Ideographic Supplement: U+1F200–U+1F2FF" in *Section 22.10, Enclosed and Square*.

18.1.14 Kangxi Radicals and CJK Radicals Supplement: U+2F00–U+2FD5, U+2E80–U+2EF3

The Unicode Standard includes two blocks of Han ideographic radicals that are commonly used to index ideograph dictionaries: the Kangxi Radicals block (U+2F00..U+2FD5), which contains the 214 radicals as used in the eighteenth-century *Kangxi* dictionary, and the CJK Radicals Supplement block (U+2E80..U+2EF3), which contains variant forms of some Kangxi radicals, either when they occur as ideograph components or in simplified form according to conventions in China and Japan.

The term *radical* comes from the Latin *radix*, which means "root," and refers to the part of an ideograph under which it is classified in most ideograph dictionaries. See "*Radical-Stroke Indices*" earlier in this section for a more detailed discussion of how ideographic radicals are used in radical-stroke indices.

Nearly all of the characters in the Kangxi Radicals and CJK Radicals Supplement blocks are equivalent to ideographs in the CJK Unified Ideographs blocks, but should not be used interchangeably. (See the "*Semantics*" subsection below.) Radicals that have one form as an independent ideograph and another as part of an ideograph are generally encoded in both forms in the CJK Unified Ideographs blocks, such as U+6C34 水 and U+6C35 氵 for the radical meaning "water." See the Equivalent_Unified_Ideograph property in the Unicode Character Database for mappings of nearly all characters in these blocks to equivalent ideographs in the CJK Unified Ideographs blocks.

Standards. CNS 11643-1992 included a block of radicals separate from its ideograph block, which included 213 of the 214 Kangxi radicals. The missing radical is the 34th one, which is encoded as U+2F21 夊 KANGXI RADICAL GO in the Unicode Standard. Amendment 1 of the CNS 11643:2007 standard, which was published in 2023, appended the missing radical to this block, which now includes all 214 Kangxi radicals.

Chinese and Non-Chinese Simplified Radicals. Chinese is not the only language whose writing system uses simplified radicals. Japanese, and to some extent Vietnamese, also make use of simplified radicals. Among the simplified radicals, a small number are shared by Chinese and non-Chinese languages, such as U+2EA6 冄 CJK RADICAL SIMPLIFIED HALF TREE TRUNK and U+2EE8 麦 CJK RADICAL SIMPLIFIED WHEAT. Others have separate Chinese and Japanese forms, such as U+2EEE 齿 CJK RADICAL C-SIMPLIFIED TOOTH and U+2EED 歯 CJK RADICAL J-SIMPLIFIED TOOTH. Some simplified radicals are not included in the CJK Radicals Supplement block, such as U+9F21 鼡, which is the Japanese simplified form of U+2FCF 鼠 KANGXI RADICAL RAT. See *Table 18-8* for a complete treatment of Chinese simplified and non-Chinese simplified radicals, together with their equivalent unified ideographs.

Table 18-8. Chinese Simplified Versus Non-Chinese Simplified
Radicals

Radical	Traditional Form		Chinese Simplified Form		Non-Chinese Simplified Form	
182	U+2FB5 風	U+98A8 風	U+2EDB 风	U+98CE 风	U+322C4 凩	
208	U+2FCF 鼠	U+9F20 鼠			U+9F21 鼡	
210	U+2FD1 齊	U+9F4A 齊	U+2EEC 齐	U+9F50 齐	U+2EEB 斉	U+6589 斉
211	U+2FD2 齒	U+9F52 齒	U+2EEE 齿	U+9F7F 齿	U+2EED 歯	U+6B6F 歯
212	U+2FD3 龍	U+9F8D 龍	U+2EF0 龙	U+9F99 龙	U+2EEF 竜	U+7ADC 竜
					U+31DE5 竜	
213	U+2FD4 龜	U+9F9C 龜	U+2EF3 龟	U+9F9F 龟	U+2EF2 亀	U+4E80 亀

Semantics. Characters in the CJK Radicals Supplement and Kangxi Radicals blocks
should not be used as ideographs, because they have different properties and seman-
tics. For example, U+2F00 一 KANGXI RADICAL ONE should not be used in lieu of
U+4E00 一 CJK UNIFIED IDEOGRAPH-4E00. The former is to be treated as a symbol,
and the latter is to be treated as a word or a part of a word. Except in circumstances
where it is necessary to make a semantic distinction between an ideograph in its role
as a radical and the same ideograph in its role as an ideograph, the characters in the
CJK Unified Ideographs blocks should be used instead of the characters in these
blocks.

Representative Glyphs. The Kangxi Radicals block uses representative glyphs that
closely adhere to the forms as found in the *Kangxi* dictionary itself, which are inde-
pendent of any particular regional convention. However, the CJK Radicals Supple-
ment block includes regional variants whose representative glyphs are appropriate
for the region. For example, U+2EEB 斉 CJK RADICAL J-SIMPLIFIED EVEN and
U+2EEF 竜 CJK RADICAL J-SIMPLIFIED DRAGON adhere to conventions as used in
Japan.

18.1.15 CJK Additions from HKSCS and GB 18030

Several characters have been encoded because of developments in HKSCS-2001 (the
Hong Kong Supplementary Character Set) and GB 18030-2000 (the PRC National
Standard). Both of these encoding standards were published with mappings to
Unicode Private Use Area code points. PUA ideographic characters that could not be
remapped to non-PUA CJK ideographs were added to the existing block of CJK
Unified Ideographs. Fourteen new ideographs (U+9FA6..U+9FB3) were added from
HKSCS, and eight multistroke ideographic components (U+9FB4..U+9FBB) were
added from GB 18030.

To complete the mapping to these two Chinese standards, a number of non-ideo-
graphic characters were encoded elsewhere in the standard. In particular, two symbol
characters from HKSCS were added to the existing Miscellaneous Technical block:

U+23DA EARTH GROUND and U+23DB FUSE. A new block, CJK Strokes (U+31C0..U+31EF), was created and populated with a number of stroke symbols from HKSCS. Another block, Vertical Forms (U+FE10..U+FE1F), was created for vertical punctuation compatibility characters from GB 18030.

18.1.16 CJK Strokes: U+31C0–U+31EF

Characters in the CJK Strokes block are single-stroke components of CJK ideographs. The first characters assigned to this block were 16 HKSCS–2001 PUA characters that had been excluded from CJK Unified Ideograph Extension B on the grounds that they were not true ideographs. Further additions consist of traditionally defined stroke types attested in the representative forms appearing in the Unicode CJK ideograph code charts or occurring in pre-unification source glyphs. See the Equivalent_Unified_Ideograph property in the Unicode Character Database for mappings of most CJK strokes to equivalent CJK unified ideographs.

CJK strokes are used with highly specific semantics (primarily to index ideographs), but they may lack the monosyllabic pronunciations and logographic functions typically associated with independent ideographs. The strokes in this block are single strokes of well-defined types. For more information about these strokes, see *Appendix F, Documentation of CJK Strokes*.

18.1.17 Ideographic Symbols and Punctuation: U+16FE0–U+16FFF

The Ideographic Symbols and Punctuation block covers historic and less common symbols and punctuation associated with various ideographic scripts. Included, for example, are iteration marks for Tangut, Nüshu, and old Chinese, as well as reading marks associated with Vietnamese use of Han characters.

18.2 Ideographic Description Characters

18.2.1 Ideographic Description Characters: U+2FF0–U+2FFF

Although the Unicode Standard includes nearly 100,000 CJK unified ideographs, thousands of extremely rare CJK ideographs have nevertheless been left unencoded. Research into cataloging additional ideographs for encoding continues, but it is anticipated that at no point will the entire set of potential, encodable ideographs be completely exhausted. In particular, ideographs continue to be coined and such new coinages will invariably be unencoded.

The 16 characters in the Ideographic Description Characters block plus the additional Ideographic Description character encoded at U+31EF provide a mechanism for the standard interchange of text that must reference unencoded ideographs. Unencoded ideographs can be described using these characters and encoded ideographs; the reader can then create a mental picture of the ideographs from the description.

This process is different from a formal *encoding* of an ideograph. There is no canonical description of unencoded ideographs; there is no semantic assigned to described ideographs; there is no equivalence defined for described ideographs. Conceptually, ideographic descriptions are more akin to the English phrase "an 'e' with an acute accent on it" than to the character sequence <U+0065, U+0301>.

In particular, support for the characters in the Ideographic Description Characters block does *not* require the rendering engine to recreate the graphic appearance of the described character.

Note also that many of the ideographs that users might represent using the Ideographic Description characters will be formally encoded in future versions of the Unicode Standard.

The Ideographic Description Algorithm depends on the fact that virtually all CJK ideographs can be broken down into smaller pieces that are themselves ideographs. The broad coverage of the ideographs already encoded in the Unicode Standard implies that the vast majority of unencoded ideographs can be represented using the Ideographic Description characters.

Although Ideographic Description Sequences are intended primarily to represent unencoded ideographs and should not be used in data interchange to represent encoded ideographs, they also have pedagogical and analytic uses. A researcher, for example, may choose to represent the character U+86D9 蛙 as "⿰虫圭" in a database to provide a link between it and other characters sharing its phonetic, such as U+5A03 娃. The IRG is using Ideographic Description Sequences in this fashion to

help provide a first-approximation, machine-generated set of unifications for its current work.

Applicability to Other Scripts. The characters in the Ideographic Description Characters block were originally derived from a Chinese standard and were encoded for use specifically in describing CJK ideographs. As a result, the following detailed description of Ideographic Description Sequences is specified entirely in terms of CJK unified ideographs and CJK radicals. However, there are several large, historic East Asian scripts whose writing systems were heavily influenced by the Han script. Like the Han script, those siniform historic scripts, which include Tangut, Jurchen, and Khitan, are logographic in nature. Furthermore, they built up characters using radicals and components, and with side-by-side and top-to-bottom stacking very similar in structure to the way CJK ideographs are composed.

The general usefulness of Ideographic Description Sequences for describing unencoded characters and the applicability of the characters in the Ideographic Description Characters block to description of siniform logographs mean that the syntax for Ideographic Description Sequences can be generalized to extend to additional East Asian logographic scripts.

Ideographic Description Sequences. Ideographic Description Sequences are defined by the following grammar. The list of characters associated with the *Ideographic* and *Radical* properties can be found in the Unicode Character Database. In particular, the *Ideographic* property is intended to apply to other siniform ideographic systems, in addition to CJK ideographs. Nüshu ideographs, Tangut ideographs, and Tangut components can also be used as elements of an Ideographic Description Sequence.

```
IDS := Ideographic | Radical | CJK_Stroke | Private Use | U+FF1F
   | IDS_UnaryOperator IDS
   | IDS_BinaryOperator IDS IDS
   | IDS_TrinaryOperator IDS IDS IDS
CJK_Stroke := U+31C0 | ... | U+31E5
IDS_UnaryOperator := U+2FFE | U+2FFF
IDS_BinaryOperator := U+2FF0 | U+2FF1 | U+2FF4 | ... | U+2FFD
   | U+31EF
IDS_TrinaryOperator := U+2FF2 | U+2FF3
```

Previous versions of the Unicode Standard imposed various limits on the length of a sequence or parts of it, and restricted the use of IDSes to CJK Unified Ideographs. Those limits and restrictions are no longer imposed by the standard. Although not formally proscribed by the syntax, it is not a good idea to mix scripts in any given Ideographic Description Sequence. For example, it is not meaningful to mix CJK ideographs or CJK radicals with Tangut ideographs or components in a single description.

The operators indicate the relative graphic positions of the operands running from left to right, from top to bottom, or from enclosure to enclosed. A user wishing to represent an unencoded ideograph will need to analyze its structure to determine how to describe it using an Ideographic Description Sequence. As a rule, it is best to use the natural radical-phonetic division for an ideograph if it has one and to use as short a description sequence as possible; however, there is no requirement that these rules be followed. Beyond that, the shortest possible Ideographic Description Sequence is preferred.

Figure 18-8 provides an example IDS for each of the IDCs, along with annotated versions of the IDCs that indicate the order of their operands.

Figure 18-8. Examples of Ideographic Description Characters

U+2FF0	U+4EC1	仁	→	亻 二
U+2FF1	U+5409	吉	→	士 口
U+2FF2	U+8857	街	→	彳 圭 亍
U+2FF3	U+58F9	壹	→	士 冖 豆
U+2FF4	U+56DE	回	→	口 口
U+2FF5	U+51F0	凰	→	几 皇
U+2FF6	U+51F6	凶	→	凵 乂
U+2FF7	U+5321	匡	→	匚 王
U+2FF8	U+4EC4	仄	→	厂 人
U+2FF9	U+5F0F	式	→	弋 工
U+2FFA	U+8D85	超	→	走 召
U+2FFB	U+5DEB	巫	→	工 从
U+2FFC	U+355A	叉	→	又 丶
U+2FFD	U+6C37	氷	→	水 丶
U+2FFE	U+23944	㽝	→	正
U+2FFF	U+20114	㐸	→	予
U+31EF	U+2002A	其		其 丿
	U+5187	冇	→	有 二

In contrast to the other IDCs, most of which are used to combine components, U+31EF IDEOGRAPHIC DESCRIPTION CHARACTER SUBTRACTION is used to describe the removal (or "subtraction") of a stroke (or more complex component) from a target character. Its first argument is the ideograph (or component) from which a piece is to be deleted, and the second argument is the stroke (or component) that is to

be removed. If the target character lacks the stroke or component to be removed, the sequence has no meaning. The typical use case for U+31EF would be in describing the many historical instances of Han naming taboo characters that exhibit removal of a stroke in the character to avoid the given name of an emperor or an emperor's ancestor. It might also be used to describe modern neologisms, such as the characters for 乒乓 *pīngpāng*, derived by removal of one stroke each from 兵.

Figure 18-9 illustrates the use of the IDS grammar to provide descriptions of encoded or unencoded ideographs. Examples 9 through 14 illustrate more complex Ideographic Description Sequences showing the use of some of the less common operators.

Figure 18-9. Using the Ideographic Description Characters
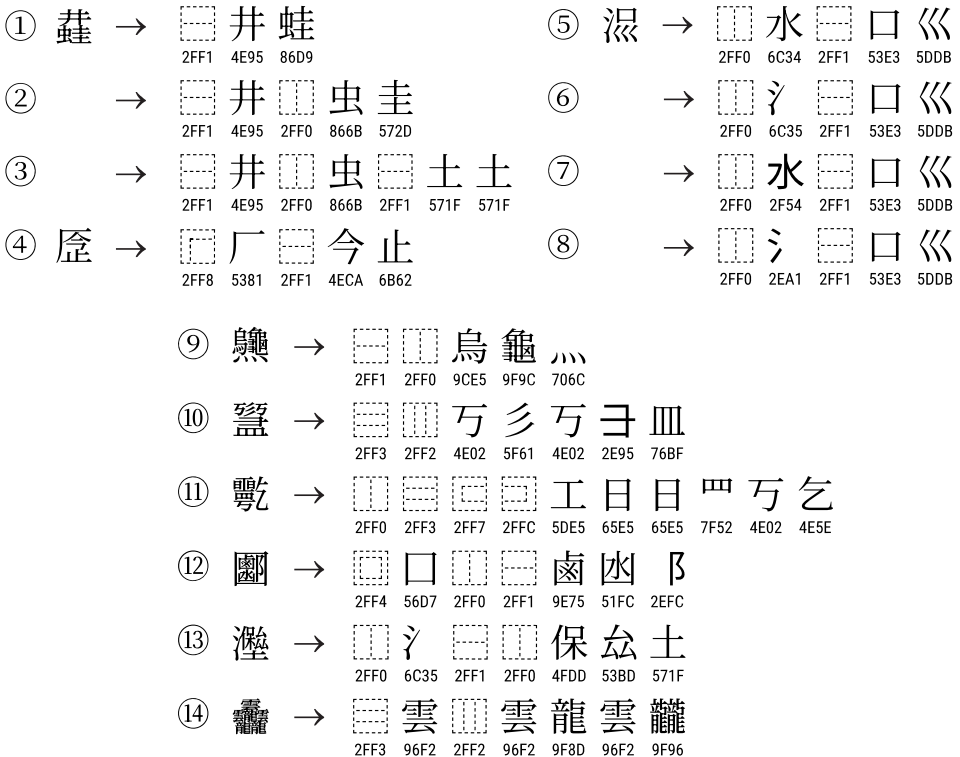
Equivalence. Many unencoded ideographs can be described in more than one way using this algorithm, either because the pieces of a description can themselves be broken down further (examples 1 through 3 in *Figure 18-9*) or because duplications appear within the Unicode Standard (examples 5 through 8 in *Figure 18-9*).

The Unicode Standard does not define equivalence for two Ideographic Description Sequences that are not identical. *Figure 18-9* contains numerous examples illustrating how different Ideographic Description Sequences might be used to describe the same ideograph.

In particular, Ideographic Description Sequences should not be used to provide alternative graphic representations of encoded ideographs in data interchange. Searching, collation, and other content-based text operations would then fail.

Interaction with the Ideographic Variation Mark. U+303E IDEOGRAPHIC VARIATION INDICATOR (IVI) normally occurs before a CJK unified ideograph, but it may also be placed before an Ideographic Description Sequence to indicate that the description is merely an approximation of the ideograph desired. The IVI is not considered a part of the Ideographic Description Sequence and does not invalidate the sequence.

Rendering. Ideographic Description characters are visible characters and are not to be treated as control characters. Thus the sequence U+2FF1 U+4E95 U+86D9 must have a distinct appearance from U+4E95 U+86D9.

An implementation may render a valid Ideographic Description Sequence either by rendering the individual characters separately or by parsing the Ideographic Description Sequence and drawing the ideograph so described. In the latter case, the Ideographic Description Sequence should be treated as a ligature of the individual characters for purposes of hit testing, cursor movement, and other user interface operations. (See *Section 5.11, Editing and Selection*.)

Character Boundaries. Ideographic Description characters are not combining characters, and there is no requirement that they affect character or word boundaries. Thus U+2FF1 U+4E95 U+86D9 may be treated as a sequence of three characters or even three words.

Implementations of the Unicode Standard may choose to parse Ideographic Description Sequences when calculating word and character boundaries. Note that such a decision will make the algorithms involved significantly more complicated and slower.

Standards. Most of the Ideographic Description characters are found in GBK—an extension to GB 2312-80 that added all 20,902 Unicode Version 1.1 ideographs not already in GB 2312-80. GBK is defined as a normative annex of GB 13000.1-93.

18.3 Bopomofo

18.3.1 Bopomofo: U+3100–U+312F, U+31A0–U+31BF

Bopomofo constitute a set of characters used to annotate or teach the phonetics of Chinese, primarily the standard Mandarin language. These characters are used in dictionaries and teaching materials, but not in the actual writing of Chinese text. The formal Chinese names for this alphabet are *Zhuyin-Zimu* ("phonetic alphabet") and *Zhuyin-Fuhao* ("phonetic symbols"), but the informal term "Bopomofo" (analogous to "ABCs") provides a more serviceable English name and is also used in China. The Bopomofo were developed as part of a populist literacy campaign following the 1911 revolution; thus they are acceptable to all branches of modern Chinese culture, although in the People's Republic of China their function has been largely taken over by the Pinyin romanization system.

Bopomofo is a hybrid writing system—part alphabet and part syllabary. The letters of Bopomofo are used to represent either the initial parts or the final parts of a Chinese syllable. The initials are just consonants, as for an alphabet. The finals constitute either simple vowels, vocalic diphthongs, or vowels plus nasal consonant combinations. Because a number of Chinese syllables have no initial consonant, the Bopomofo letters for finals may constitute an entire syllable by themselves. More typically, a Chinese syllable is represented by one initial consonant letter, followed by one final letter. In some instances, a third letter is used to indicate a complex vowel nucleus for the syllable. For example, the syllable that would be written *luan* in Pinyin is segmented l-u-an in Bopomofo—that is, <U+310C, U+3128, U+3122>.

Standards. The standard Mandarin set of Bopomofo is included in the People's Republic of China standards GB 2312 and GB 18030, and in the Republic of China (Taiwan) standard CNS 11643.

Mandarin Tone Marks. Small modifier letters used to indicate the five Mandarin tones are part of the Bopomofo system. In the Unicode Standard they have been unified into the Modifier Letter range, as shown in *Table 18-9*.

Table 18-9. Mandarin Tone Marks

first tone	U+02C9 MODIFIER LETTER MACRON
second tone	U+02CA MODIFIER LETTER ACUTE ACCENT
third tone	U+02C7 CARON
fourth tone	U+02CB MODIFIER LETTER GRAVE ACCENT
light tone	U+02D9 DOT ABOVE

Standard Mandarin Bopomofo. The order of the Mandarin Bopomofo letters U+3105.. U+3129 is standard worldwide. The code offset of the first letter U+3105

BOPOMOFO LETTER B from a multiple of 16 is included to match the offset in the ISO-registered standard GB 2312.

Extended Bopomofo. To represent the sounds of Chinese dialects other than Mandarin, the basic Bopomofo set U+3105..U+3129 has been augmented by additional phonetic characters. These extensions are much less broadly recognized than the basic Mandarin set. The three extended Bopomofo characters U+312A..U+312C are cited in some standard reference works, such as the encyclopedia *Xin Ci Hai*. Another set of 24 extended Bopomofo, encoded at U+31A0..U+31B7, was designed in 1948 to cover additional sounds of the Minnan and Hakka dialects. The extensions are used together with the main set of Bopomofo characters to provide a complete phonetic orthography for those dialects. The four characters encoded at U+31BC..U+31BF were designed to represent additional sounds found in Cantonese.

The small characters encoded at U+31B4..U+31B7 and U+31BB represent syllable-final consonants not present in standard Mandarin or in Mandarin dialects. They have the same shapes as Bopomofo "b", "d", "k", "h", and "g," respectively, but are rendered in a smaller form than the initial consonants; they are also generally shown close to the syllable medial vowel character. These final letters are encoded separately so that the Minnan and Hakka dialects can be represented unambiguously in plain text without having to resort to subscripting or other fancy text mechanisms to represent the final consonants. In Cantonese, final consonants not covered by the set of standard Bopomofo rhymes ending in *-n* or *-ng* are instead represented by full-sized letters for "p", "t", "k", "m", "n", "ng".

Three Bopomofo letters for sounds found in non-Chinese languages are encoded in the range U+31B8..U+31BA. These characters are used in the Hmu and Ge languages, members of the Hmong-Mien (or Miao-Yao) language family, spoken primarily in southeastern Guizhou. The characters are part of an obsolete orthography for Hmu and Ge devised by the missionary Maurice Hutton in the 1920s and 1930s. A small group of Hmu Christians are still using a hymnal text written by Hutton that contains these characters.

U+312E ㄮ BOPOMOFO LETTER O WITH DOT ABOVE, which was initially thought to be a CJK Unified Ideograph because it appears in Japan's *Dai Kan-Wa Jiten* as a kanji, is the original form of U+311C ㄜ BOPOMOFO LETTER E. The Mandarin sound "e" was originally written as U+311B ㄛ BOPOMOFO LETTER O with a dot above. This dotted form was later replaced by a new character that uses a vertical stroke instead of a dot, which is U+311C ㄜ BOPOMOFO LETTER E.

Extended Bopomofo Tone Marks. In addition to the Mandarin tone marks enumerated in *Table 18-9*, other tone marks appropriate for use with the extended Bopomofo transcriptions of Minnan and Hakka can be found in the Modifier Letter range, as shown in *Table 18-10*. The "departing tone" refers to the *qusheng* in traditional Chinese tonal analysis, with the *yin* variant historically derived from voiceless initials

and the *yang* variant from voiced initials. Southern Chinese dialects in general maintain more tonal distinctions than Mandarin does.

Table 18-10. Minnan and Hakka Tone Marks

yin departing tone	U+02EA MODIFIER LETTER YIN DEPARTING TONE MARK
yang departing tone	U+02EB MODIFIER LETTER YANG DEPARTING TONE MARK

Rendering of Bopomofo. Bopomofo is rendered from left to right in horizontal text, but also commonly appears in vertical text. It may be used by itself in either orientation, but typically appears in interlinear annotation of Chinese (Han character) text. Children's books are often completely annotated with Bopomofo pronunciations for every character. This interlinear annotation is structurally quite similar to the system of Japanese *ruby* annotation, but it has additional complications that result from the explicit usage of tone marks with the Bopomofo letters.

U+3127 BOPOMOFO LETTER I has notable variation in rendering in horizontal and vertical layout contexts. In traditional typesetting, the stroke of the glyph was chosen to stand perpendicular to the writing direction. In that practice, the glyph is shown as a horizontal stroke in vertically set text, and as a vertical stroke in horizontally set text. However, modern digital typography has changed this practice. All modern fonts use a horizontal stroke glyph for U+3127, and that form is generally used in both horizontal and vertical layout contexts. In the Unicode Standard, the form in the charts follows the modern practice, showing a horizontal stroke for the glyph; the vertical stroke form is considered to be an occasionally occurring variant. Earlier versions of the standard followed traditional typographic practice, and showed a vertical stroke glyph in the charts.

In horizontal interlineation, the Bopomofo is generally placed above the corresponding Han character(s); tone marks, if present, appear at the end of each syllabic group of Bopomofo letters. In vertical interlineation, the Bopomofo is generally placed on the right side of the corresponding Han character(s); tone marks, if present, appear in a separate interlinear row to the right side of the vowel letter. When using extended Bopomofo for Minnan and Hakka, the tone marks may also be mixed with European digits 0–9 to express changes in actual tonetic values resulting from juxtaposition of basic tones.

18.4 Hiragana and Katakana

18.4.1 Hiragana: U+3040–U+309F

Hiragana is the cursive syllabary used to write Japanese words phonetically and to write sentence particles and inflectional endings. It is also commonly used to indicate the pronunciation of Japanese words. Hiragana syllables are phonetically equivalent to the corresponding Katakana syllables.

Standards. The Hiragana block is based on the JIS X 0208-1990 standard, extended by the nonstandard syllable U+3094 HIRAGANA LETTER VU, which is included in some Japanese corporate standards. Some additions are based on the JIS X 0213:2000 standard.

Combining Marks. Hiragana and the related script Katakana use U+3099 COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK and U+309A COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK to generate voiced and semivoiced syllables from the base syllables, respectively. All common precomposed combinations of base syllable forms using these marks are already encoded as characters, and use of these precomposed forms is the predominant JIS usage. These combining marks must follow the base character to which they apply. Because most implementations and JIS standards treat these marks as spacing characters, the Unicode Standard contains two corresponding noncombining (spacing) marks at U+309B and U+309C.

Iteration Marks. The two characters U+309D HIRAGANA ITERATION MARK and U+309E HIRAGANA VOICED ITERATION MARK are punctuation-like characters that denote the iteration (repetition) of a previous syllable according to whether the repeated syllable has an unvoiced or voiced consonant, respectively.

Vertical Text Digraph. U+309F HIRAGANA DIGRAPH YORI is a digraph form which was historically used in vertical display contexts, but which is now also found in horizontal layout.

18.4.2 Katakana: U+30A0–U+30FF

Katakana is the noncursive syllabary used to write non-Japanese (usually Western) words phonetically in Japanese. It is also used to write Japanese words with visual emphasis. Katakana syllables are phonetically equivalent to corresponding Hiragana syllables. Katakana contains two characters, U+30F5 KATAKANA LETTER SMALL KA and U+30F6 KATAKANA LETTER SMALL KE, that are used in special Japanese spelling conventions (for example, the spelling of place names that include archaic Japanese connective particles).

Standards. The Katakana block is based on the JIS X 0208-1990 standard. Some additions are based on the JIS X 0213:2000 standard.

Punctuation-like Characters. U+30FB KATAKANA MIDDLE DOT is used to separate words when writing non-Japanese phrases. U+30A0 KATAKANA-HIRAGANA DOUBLE HYPHEN is a delimiter occasionally used in analyzed Katakana or Hiragana textual material.

U+30FC KATAKANA-HIRAGANA PROLONGED SOUND MARK is used predominantly with Katakana and occasionally with Hiragana to denote a lengthened vowel of the previously written syllable. The two iteration marks, U+30FD KATAKANA ITERATION MARK and U+30FE KATAKANA VOICED ITERATION MARK, serve the same function in Katakana writing that the two Hiragana iteration marks serve in Hiragana writing.

Vertical Text Digraph. U+30FF KATAKANA DIGRAPH KOTO is a digraph form which was historically used in vertical display contexts, but which is now also found in horizontal layout.

18.4.3 Katakana Phonetic Extensions: U+31F0–U+31FF

These extensions to the Katakana syllabary are all "small" variants. They are used in Japan for phonetic transcription of Ainu and other languages. They may be used in combination with U+3099 COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK and U+309A COMBINING KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK to indicate modification of the sounds represented.

Standards. The Katakana Phonetic Extensions block is based on the JIS X 0213:2000 standard.

18.4.4 Small Kana Extension: U+1B130-U+1B16F

The Small Kana Extension block contains additional small variants for the Hiragana syllabary and the Katakana syllabary. A significant number of these small variant kana are attested from sources, which include phonetic transcription of non-Japanese terms in musical scores, maps showing place names, and other documents. The small kana variants currently included in this block cover the best attested subset, including forms used in Old Japanese. They are ordered so that gaps in the code chart may be filled in with further small variant kana, when their attestations are better documented.

18.4.5 Kana Supplement: U+1B000–U+1B0FF

Kana Extended-A: U+1B100–U+1B12F

The Kana Supplement and Kana Extended-A blocks are intended for the encoding of historic and variant forms of Japanese kana characters, including those variants collectively known as *hentaigana* (variant shaped kana) in Japanese.

The character U+1B000 KATAKANA LETTER ARCHAIC E is an obsolete form of U+30A8 KATAKANA LETTER E, which has not been used in Japanese orthography for about one thousand years. In its pre-10th century use, this character represented the syllable "e", and U+30A8 KATAKANA LETTER E represented the syllable "ye". The character U+1B001 HIRAGANA LETTER ARCHAIC YE was originally encoded to represent a long-obsolete syllable that would have come between U+3086 HIRAGANA LETTER YU and U+3088 HIRAGANA LETTER YO. This syllable merged with "e", which is now represented by U+3048 HIRAGANA LETTER E. These relationships are illustrated in *Figure 18-10*.

Figure 18-10. Japanese Historic Kana for *e* and *ye*

The hentaigana 𛀁, which would have been named HENTAIGANA LETTER E-1, has been unified with the existing U+1B001 HIRAGANA LETTER ARCHAIC YE and is aliased accordingly. When sorting, U+1B001 HIRAGANA LETTER ARCHAIC YE should appear between U+1B00E HENTAIGANA LETTER U-5 and U+1B00F HENTAIGANA LETTER E-2.

The 285 remaining characters in these blocks are additional hentaigana that represent obsolete or nonstandard hiragana that were in use in Japan up until the script reform of 1900 that standardized the use of a single character for each syllable. Hentaigana are still in use today in Japan, but are limited to Japan's family registry (*koseki* in Japanese) and specialized uses, such as business signage and other decor that are specifically designed to convey a feeling of nostalgia or traditional charm.

Each hentaigana is associated with a single parent unified ideograph, a cursive form of which served as the basis for its shape, and generally correspond to a single syllable. Hentaigana that correspond to the same syllable, but that do not share the same parent unified ideograph have different shapes and are therefore encoded separately. For example, U+1B006 HENTAIGANA LETTER I-1 through U+1B009

HENTAIGANA LETTER I-4 all correspond to the same syllable *i* (U+3044 HIRAGANA LETTER I), but have parent unified ideographs U+4EE5 以, U+4F0A 伊, U+610F 意, and U+79FB 移, respectively, as shown in *Figure 18-11*.

Figure 18-11. Hentaigana Distinct Parent Ideographs

Some hentaigana that correspond to the same syllable and share the same parent unified ideograph are also encoded separately because they have different shapes. For example, U+1B080 HENTAIGANA LETTER NA-3 through U+1B082 HENTAIGANA LETTER NA-5 correspond to the same syllable *na* (U+306A HIRAGANA LETTER NA) and share the same parent unified ideograph U+5948 奈, as shown in *Figure 18-12*.

Figure 18-12. Other Hentaigana Examples

A small number of hentaigana that share the same parent unified ideograph are associated with two or three different syllables reflected in their names, such as U+1B07D HENTAIGANA LETTER TO-RA that is associated with the syllables *to* (U+3068 HIRAGANA LETTER TO) and *ra* (U+3089 HIRAGANA LETTER RA), and U+1B11D HENTAIGANA LETTER N-MU-MO-1 that is associated with the syllables *n* (U+3093 HIRAGANA LETTER N), *mu* (U+3080 HIRAGANA LETTER MU), and *mo* (U+3082 HIRAGANA LETTER MO). Their parent unified ideographs are U+7B49 等 and U+65E0 无, respectively. These associations are also illustrated in *Figure 18-12*.

18.4.6 Kana Extended-B: U+1AFF0–U+1AFFF

The Kana Extended-B block encodes tone marks used alongside *furigana* to annotate Minnan languages in an orthography known in Japanese as Taiwanese kana (台湾語

仮名, *taiwango kana*). These character forms date back to the work of the Japanese linguist Naoyoshi Ogawa (小川尚義) in the early 20th century.

These characters are not, however, a mere historical curiosity. The linguist Âng Ûi-jîn (洪惟仁) produced a dictionary using them as recently as 1993, the *Tâi-jı̍t Tōa Sû-tián* (臺日大辭典).

The kana with their tone marks appear historically as interlinear annotations to the right of each ideographic character in vertical text. They are not historically attested in horizontally typeset documents. The tone marks in this block appear to the right of the kana characters, in some ways similar to the rendering of tone marks with Bopomofo characters, described in *Section 18.3, Bopomofo*. At most one tone mark from this block appears to the right of each syllabic group. Marks from other blocks may also appear above or below the kana.

The orthography contains two diacritics, which represent various sound changes depending on the Minnan language being annotated. U+0323 COMBINING DOT BELOW is used to represent the aspiration mark (送氣符, *sàng-khì hû*). U+0305 COMBINING OVERLINE is used to represent the line above which, depending on the dialect of Minnan being annotated, results in various sound changes (発音符, *huat-im hû*). COMBINING OVERLINE may occur over both the small and large versions of the vowels ウ and オ when the Quanzhou dialect (泉州話, *Choân-chiu-oē*) is being annotated.

Figure 18-13 shows an example with the annotated Minnan phrase 恬恬聽, which means "quietly listening," typeset in vertical interlineation. Such interlinear text cannot be represented directly in plain text; higher level protocols must render the ideographic block characters and the *furigana* in separate runs. For this and subsequent examples, the CJK ideograph sequence is <606C 606C 807D>. The *furigana* annotation sequence in each case is <30C1 0305 30A1 30E0 1AFF5 30C1 0305 30A1 30E0 1AFF5 30C1 0305 0323 30A2 1AFF7>. The dialect of Minnan affects the annotation, so this is but one possible annotation of 恬恬聽, from a Taiwanese textbook for teaching Japanese published in 1902.

Figure 18-13. Vertical Layout with Interlineation

In non-interlinear vertical katakana text, the tone marks once again appear to the right side of the katakana, as shown in *Figure 18-14*. Historically, they were most often used for this purpose in pedagogical materials.

Figure 18-**14.** Vertical Layout without Interlineation

チァム｜ チァム｜ チ・ア

As most modern CJK documents are horizontally typeset, it may be convenient to include these *furigana* in horizontal interlineation. However, as there are neither historic nor widely accepted forms of the tone mark characters when displayed above ideographic characters, rather than to their right, the *furigana* may be rendered as if the text were vertical, but with the ideographic characters being written in horizontal order, as in *Figure 18-15*.

Figure 18-**15.** Horizontal Layout with Interlineation

恬 チァム｜ 恬 チァム｜ 聽 チ・ア

In non-interlinear horizontal text the recommended presentation is to display the tone marks after the katakana syllables, as shown in *Figure 18-16*. Horizontal text which uses Kana Extended-B characters is ahistorical, but still extant, as modern CJK languages are often written horizontally.

Figure 18-**16.** Horizontal Layout without Interlineation

チァム｜チァム｜チ・ア

The characters of the Kana Extended-B block only annotate regular, fullwidth katakana characters. There are no historical examples of the annotation of halfwidth forms of katakana found in the block Halfwidth and Fullwidth Forms.

18.5 Halfwidth and Fullwidth Forms

18.5.1 Halfwidth and Fullwidth Forms: U+FF00–U+FFEF

In the context of East Asian coding systems, a double-byte character set (DBCS), such as JIS X 0208-1990 or KS X 1001:1998, is generally used together with a single-byte character set (SBCS), such as ASCII or a variant of ASCII. Text that is encoded with both a DBCS and SBCS is typically displayed such that the glyphs representing DBCS characters occupy two display cells—where a display cell is defined in terms of the glyphs used to display the SBCS (ASCII) characters. In these systems, the two-display-cell width is known as the *fullwidth* or *zenkaku* form, and the one-display-cell width is known as the *halfwidth* or *hankaku* form. While *zenkaku* and *hankaku* are Japanese terms, the display-width concepts apply equally to Korean and Chinese implementations.

Because of this mixture of display widths, certain characters often appear twice—once in fullwidth form in the DBCS repertoire and once in halfwidth form in the SBCS repertoire. To achieve round-trip conversion compatibility with such mixed-width encoding systems, it is necessary to encode both fullwidth and halfwidth forms of certain characters. This block consists of the additional forms needed to support conversion for existing texts that employ both forms.

In the context of conversion to and from such mixed-width encodings, all characters in the General Scripts Area should be construed as halfwidth (*hankaku*) characters if they have a fullwidth equivalent elsewhere in the standard or if they do not occur in the mixed-width encoding; otherwise, they should be construed as fullwidth (*zenkaku*). Specifically, most characters in the CJK Miscellaneous Area and the CJKV Ideograph Area, along with the characters in the CJK Compatibility Ideographs, CJK Compatibility Forms, and Small Form Variants blocks, should be construed as fullwidth (*zenkaku*) characters. For a complete description of the East Asian Width property, see Unicode Standard Annex #11, "East Asian Width."

The characters in this block consist of fullwidth forms of the ASCII block (except SPACE), certain characters of the Latin-1 Supplement, and some currency symbols. In addition, this block contains halfwidth forms of the Katakana and Hangul Compatibility Jamo characters. Finally, a number of symbol characters are replicated here (U+FFE8..U+FFEE) with explicit halfwidth semantics.

Unifications. The fullwidth form of U+0020 SPACE is unified with U+3000 IDEO-GRAPHIC SPACE.

18.6 Hangul

Korean Hangul may be considered a featural syllabic script. As opposed to many other syllabic scripts, the syllables are formed from a set of alphabetic components in a regular fashion. These alphabetic components are called *jamo*.

The name *Hangul* itself is just one of several terms that may be used to refer to the script. In some contexts, the preferred term is simply the generic *Korean characters*. *Hangul* is used more frequently in South Korea, whereas a basically synonymous term *Choseongul* is preferred in North Korea. A politically neutral term, *Jeongum*, may also be used.

The Unicode Standard contains both the complete set of precomposed modern Hangul syllable blocks and a set of conjoining Hangul jamo. The conjoining Hangul jamo can be used to represent all of the modern Hangul syllable blocks, as well as the obsolete syllable blocks composed of at least one Hangul jamo that the Korean orthographic standard in 1933 excluded from modern use. For a description of conjoining jamo behavior and precomposed Hangul syllables, see *Section 3.12, Conjoining Jamo Behavior*. For a discussion of the interaction of combining marks with jamo and Hangul syllables, see "Combining Marks and Korean Syllables" in *Section 3.6, Combination*. Note that the representation of Old Korean requires two combining tone marks for Hangul, U+302E and U+302F.

For other blocks containing characters related to Hangul, see "Enclosed CJK Letters and Months: U+3200–U+32FF" and "CJK Compatibility: U+3300–U+33FF" in *Section 22.10, Enclosed and Square*, as well as *Section 18.5, Halfwidth and Fullwidth Forms*.

18.6.1 Hangul Jamo: U+1100–U+11FF

The Hangul Jamo block contains the most frequently used conjoining jamo. These include all of the jamo used in modern Hangul syllable blocks, as well as many of the jamo for Old Korean.

The Hangul jamo are divided into three classes: *choseong* (leading consonants, or syllable-initial characters), *jungseong* (vowels, or syllable-peak characters), and *jongseong* (trailing consonants, or syllable-final characters). Each class may, in turn, consist of one to three subunits. For example, a *choseong* syllable-initial character may either represent a single consonant sound, or a consonant cluster consisting of two or three consonant sounds. Likewise, a *jungseong* syllable-peak character may represent a simple vowel sound, or a complex diphthong or triphthong with onglide or offglide sounds. Each of these complex sequences of two or three sounds is encoded as a single conjoining jamo character. Therefore, a complete Hangul syllable

can always be conceived of as a single *choseong* followed by a single *jungseong* and (optionally) a single *jongseong*.

This block also contains two invisible filler characters which act as placeholders for a missing *choseong* or *jungseong* in an incomplete syllable. These filler characters are U+115F HANGUL CHOSEONG FILLER and U+1160 HANGUL JUNGSEONG FILLER.

18.6.2 Hangul Jamo Extended-A: U+A960–U+A97F

This block is an extension of the conjoining jamo. It contains additional complex leading consonants (*choseong*) needed to complete the set of conjoining jamo for the representation of Old Korean.

18.6.3 Hangul Jamo Extended-B: U+D7B0–U+D7FF

This block is an extension of the conjoining jamo. It contains additional complex vowels (*jungseong*) and trailing consonants (*jongseong*) needed to complete the set of conjoining jamo for the representation of Old Korean.

18.6.4 Hangul Compatibility Jamo: U+3130–U+318F

This block consists of spacing, nonconjoining Hangul consonant and vowel (jamo) elements. These characters are provided solely for compatibility with the KS X 1001:1998 standard. Unlike the characters found in the Hangul Jamo block (U+1100..U+11FF), the jamo characters in this block have no conjoining semantics.

The characters of this block are considered to be fullwidth forms in contrast with the halfwidth Hangul compatibility jamo found at U+FFA0..U+FFDF.

Standards. The Unicode Standard follows KS X 1001:1998 for Hangul Jamo elements.

Normalization. When Hangul compatibility jamo are transformed with a compatibility normalization form, NFKD or NFKC, the characters are converted to the corresponding conjoining jamo characters. Where the characters are intended to remain in separate syllables after such transformation, they may require separation from adjacent characters. This separation can be achieved by inserting any non-Korean character.

- U+200B ZERO WIDTH SPACE is recommended where the characters are to allow a line break.
- U+2060 WORD JOINER can be used where the characters are not to break across lines.

Table 18-11 illustrates how two Hangul compatibility jamo can be separated in display, even after transforming them with NFKD or NFKC.

Table 18-11. Separating Jamo Characters

Original	NFKD	NFKC	Display
ㄱ ㅏ 3131 314F	ㄱ ㅏ 1100 1161	가 AC00	가
ㄱ [ZW SP] ㅏ 3131 200B 314F	ㄱ [ZW SP] ㅏ 1100 200B 1161	ㄱ [ZW SP] ㅏ 1100 200B 1161	ㄱㅏ

18.6.5 Hangul Syllables: U+AC00–U+D7AF

The Hangul script used in the Korean writing system consists of individual conso-nant and vowel letters (jamo) that are visually combined into square display cells to form entire syllable blocks. Hangul syllables may be encoded directly as precom-posed combinations of individual jamo or as decomposed sequences of conjoining jamo.

Modern Hangul syllable blocks can be expressed with either two or three jamo, either in the form *consonant + vowel* or in the form *consonant + vowel + consonant*. There are 19 possible leading (initial) consonants (*choseong*), 21 vowels (*jungseong*), and 27 trailing (final) consonants (*jongseong*). Thus there are 399 possible two-jamo syllable blocks and 10,773 possible three-jamo syllable blocks, giving a total of 11,172 modern Hangul syllable blocks. This collection of 11,172 modern Hangul syllables encoded in this block is known as the *Johab* set.

Standards. The Hangul syllables are taken from KS C 5601-1992, representing the full Johab set. This group represents a superset of the Hangul syllables encoded in earlier versions of Korean standards (KS C 5601-1987 and KS C 5657-1991).

Equivalence. Each of the Hangul syllables encoded in this block may be represented by an equivalent sequence of conjoining jamo. The converse is not true because thou-sands of archaic Hangul syllables may be represented only as a sequence of conjoining jamo.

Hangul Syllable Composition. The Hangul syllables can be derived from conjoining jamo by a regular process of composition. The algorithm that maps a sequence of conjoining jamo to the encoding point for a Hangul syllable in the Johab set is detailed in *Section 3.12, Conjoining Jamo Behavior*.

Hangul Syllable Decomposition. Any Hangul syllable from the Johab set can be decomposed into a sequence of conjoining jamo characters. The algorithm that details the formula for decomposition is also provided in *Section 3.12, Conjoining Jamo Behavior*.

Hangul Syllable Name. The character names for Hangul syllables are derived algorithmically from the decomposition. (For full details, see *Section 3.12, Conjoining Jamo Behavior*.)

Hangul Syllable Representative Glyph. The representative glyph for a Hangul syllable can be formed from its decomposition based on the categorization of vowels shown in *Table 18-12*.

Table 18-12. Line-Based Placement of Jungseong

Vertical		Horizontal		Both	
1161	A	1169	O	116A	WA
1162	AE	116D	YO	116B	WAE
1163	YA	116E	U	116C	OE
1164	YAE	1172	YU	116F	WEO
1165	EO	1173	EU	1170	WE
1166	E			1171	WI
1167	YEO			1174	YI
1168	YE				
1175	I				

If the vowel of the syllable is based on a vertical line, place the preceding consonant to its left. If the vowel is based on a horizontal line, place the preceding consonant above it. If the vowel is based on a combination of vertical and horizontal lines, place the preceding consonant above the horizontal line and to the left of the vertical line. In either case, place a following consonant, if any, below the middle of the resulting group.

In any particular font, the exact placement, shape, and size of the components will vary according to the shapes of the other characters and the overall design of the font.

Collation. The unit of collation in Korean text is normally the Hangul syllable. The order of the syllables in the Hangul Syllables block reflects the preferred collation order used in the Republic of Korea. If sequences of Hangul syllables are collated with a simple binary comparison, the result will reflect that collation order. More sophisticated collation algorithms are required to obtain other collation orders, such as the one preferred in the Democratic People's Republic of Korea.

When Korean text includes sequences of conjoining jamo, as for Old Korean, or mixtures of precomposed syllable blocks and conjoining jamo, the easiest approach for collation is to decompose the precomposed syllable blocks into conjoining jamo before comparing. Additional steps must be taken to ensure that comparison is then done for sequences of conjoining jamo that comprise complete syllables. See Unicode

Technical Report #10, "Unicode Collation Algorithm," for more discussion about the collation of Korean.

18.7 Yi

18.7.1 Yi: U+A000–U+A4CF

The Yi syllabary encoded in Unicode is used to write the Liangshan dialect of the Yi language, a member of the Sino-Tibetan language family.

Yi is the Chinese name for one of the largest ethnic minorities in the People's Republic of China. The Yi, also known historically and in English as the Lolo, do not have a single ethnonym, but refer to themselves variously as Nuosu, Sani, Axi or Misapo. According to the 1990 census, more than 6.5 million Yi live in southwestern China in the provinces of Sichuan, Guizhou, Yunnan, and Guangxi. Smaller populations of Yi are also to be found in Myanmar, Laos, and Vietnam. Yi is one of the official languages of the PRC, with between 4 and 5 million speakers.

The Yi language is divided into six major dialects. The Northern dialect, which is also known as the Liangshan dialect because it is spoken throughout the region of the Greater and Lesser Liangshan Mountains, is the largest and linguistically most coherent of these dialects. In 1991, there were about 1.6 million speakers of the Liangshan Yi dialect. The ethnonym of speakers of the Liangshan dialect is Nuosu.

Traditional Yi Script. The traditional Yi script, historically known as Cuan or Wei, is an ideographic script. Unlike in other Chinese-influenced siniform scripts, however, the ideographs of Yi appear not to be derived from Han ideographs. One of the more widespread traditions relates that the script, comprising about 1,840 ideographs, was devised by someone named Aki during the Tang dynasty (618–907 CE). The earliest surviving examples of the Yi script are monumental inscriptions dating from about 500 years ago; the earliest example is an inscription on a bronze bell dated 1485.

There is no single unified Yi script, but rather many local script traditions that vary considerably with regard to the repertoire, shapes, and orientations of individual glyphs and the overall writing direction. The profusion of local script variants occurred largely because until modern times the Yi script was mainly used for writing religious, magical, medical, or genealogical texts that were handed down from generation to generation by the priests of individual villages, and not as a means of communication between different communities or for the general dissemination of knowledge. Although a vast number of manuscripts written in the traditional Yi script have survived to the present day, the Yi script was not widely used in printing before the 20th century.

Because the traditional Yi script is not standardized, a considerable number of glyphs are used in the various script traditions. According to one authority, there are more than 14,200 glyphs used in Yunnan, more than 8,000 in Sichuan, more than 7,000 in

Guizhou, and more than 600 in Guangxi. However, these figures are misleading—most of the glyphs are simple variants of the same abstract character. For example, a 1989 dictionary of the Guizhou Yi script contains about 8,000 individual glyphs, but excluding glyph variants reduces this count to about 1,700 basic characters, which is quite close to the figure of 1,840 characters that Aki is reputed to have devised.

Standardized Yi Script. There has never been a high level of literacy in the traditional Yi script. Usage of the traditional script has remained limited even in modern times because the traditional script does not accurately reflect the phonetic characteristics of the modern Yi language, and because it has numerous variant glyphs and differences from locality to locality.

To improve literacy in Yi, a scheme for representing the Liangshan dialect using the Latin alphabet was introduced in 1956. A standardized form of the traditional script used for writing the Liangshan Yi dialect was devised in 1974 and officially promulgated in 1980. The standardized Liangshan Yi script encoded in Unicode is suitable for writing only the Liangshan Yi dialect; it is not intended as a unified script for writing all Yi dialects. Standardized versions of other local variants of traditional Yi scripts do not yet exist.

The standardized Yi syllabary comprises 1,164 signs representing each of the allowable syllables in the Liangshan Yi dialect. There are 819 unique signs representing syllables pronounced in the high level, low falling, and midlevel tones, and 345 composite signs representing syllables pronounced in the secondary high tone. The signs for syllables in the secondary high tone consist of the sign for the corresponding syllable in the midlevel tone (or in three cases the low falling tone), plus a diacritical mark shaped like an inverted breve. For example, U+A001 YI SYLLABLE IX is the same as U+A002 YI SYLLABLE I plus a diacritical mark. In addition to the 1,164 signs representing specific syllables, a syllable iteration mark is used to indicate reduplication of the preceding syllable, which is frequently used in interrogative constructs.

Standards. In 1991, a national standard for Yi was adopted by China as GB 13134-91. This encoding includes all 1,164 Yi syllables as well as the syllable iteration mark, and is the basis for the encoding in the Unicode Standard. The syllables in the secondary high tone, which are differentiated from the corresponding syllable in the midlevel tone or the low falling tone by a diacritical mark, are not decomposable.

Naming Conventions and Order. The Yi syllables are named on the basis of the spelling of the syllable in the standard Liangshan Yi romanization introduced in 1956. The tone of the syllable is indicated by the final letter: "t" indicates the high level tone, "p" indicates the low falling tone, "x" indicates the secondary high tone, and an absence of final "t", "p", or "x" indicates the midlevel tone.

With the exception of U+A015, the Yi syllables are ordered according to their phonetic order in the Liangshan Yi romanization—that is, by initial consonant, then by vowel, and finally by tone (t, x, unmarked, and p). This is the order used in dictionaries of Liangshan Yi that are ordered phonetically.

Yi Syllable Iteration Mark. U+A015 YI SYLLABLE WU does not represent a specific syllable in the Yi language, but rather is used as a syllable iteration mark. Its character properties therefore differ from those for the rest of the Yi syllable characters. The misnomer of U+A015 as YI SYLLABLE WU derives from the fact that it is represented by the letter *w* in the romanized Yi alphabet, and from some confusion about the meaning of the gap in traditional Yi syllable charts for the hypothetical syllable "wu".

The Yi syllable iteration mark is used to replace the second occurrence of a reduplicated syllable under all circumstances. It is very common in both formal and informal Yi texts.

Punctuation. The standardized Yi script does not have any special punctuation marks, but relies on the same set of punctuation marks used for writing modern Chinese in the PRC, including U+3001 IDEOGRAPHIC COMMA and U+3002 IDEOGRAPHIC FULL STOP.

Rendering. The traditional Yi script used a variety of writing directions—for example, right-to-left in the Liangshan region of Sichuan, and top-to-bottom in columns running from left to right in Guizhou and Yunnan. The standardized Yi script follows the writing rules for Han ideographs, so characters are generally written from left to right or occasionally from top to bottom. There is no typographic interaction between individual characters of the Yi script.

Yi Radicals. To facilitate the lookup of Yi characters in dictionaries, sets of radicals modeled on Han radicals have been devised for the various Yi scripts. (For information on Han radicals, see "CJK and Kangxi Radicals" in *Section 18.1, Han*). The traditional Guizhou Yi script has 119 radicals; the traditional Liangshan Yi script has 170 radicals; and the traditional Yunnan Sani Yi script has 25 radicals. The standardized Liangshan Yi script encoded in Unicode has a set of 55 radical characters, which are encoded in the Yi Radicals block (U+A490..U+A4C5). Each radical represents a distinctive stroke element that is common to a subset of the characters encoded in the Yi Syllables block. The name used for each radical character is that of the corresponding Yi syllable closest to it in shape.

18.8 Nüshu

18.8.1 Nüshu: U+1B170–U+1B2FF

Nüshu is a siniform script devised by women to write the local Chinese dialect of Jiangyong county in the Xiaoshui Valley of southeastern Hunan province in China. Nüshu means "women's writing," and was originally used only by women, many of whom could not write Chinese Han characters. The script appeared in handwritten cloth-bound booklets of poems and songs, called San Chao Shu (三朝書), that were passed down from one "sworn sister" to another upon marriage. Nüshu also was used for other purposes, and on different media. By the late twentieth century, very few women fluent in the script were still alive. National and international attention to Nüshu has led to active efforts to study and preserve the script.

Structure. Nüshu is written vertically in columns which are laid out from right to left. Although largely based on Chinese Han characters, Nüshu characters typically represent the phonetic values of syllables, with many characters representing several homophonous words. Some signs are used as ideographs.

Names. Nüshu characters are named sequentially by prefixing the string "NUSHU CHARACTER-" to the code point. The diaeresis is not included in this prefix because of the constraints on letters that can be used in character names.

Order. The Nüshu characters are ordered by stroke count, then by vowel, consonant, and tone.

Punctuation. Nüshu has one punctuation mark, U+16FE1 NUSHU ITERATION MARK, located in the Ideographic Symbols and Punctuation block.

Sources. The Unicode Character Database contains a source data file for Nüshu called NushuSources.txt. This data file contains normative information on the source references for each Nüshu character. NushuSources.txt also contains an informative reading value for each character.

18.9 Lisu

18.9.1 Lisu: U+A4D0–U+A4FF

Somewhere between 1908 and 1914 a Karen evangelist from Myanmar by the name of Ba Thaw modified the shapes of Latin characters and created the Lisu script. Afterwards, British missionary James Outram Fraser and some Lisu pastors revised and improved the script. The script is commonly known in the West as the Fraser script. It is also sometimes called the Old Lisu script, to distinguish it from newer, Latin-based orthographies for the Lisu language.

There are 630,000 Lisu people in China, mainly in the regions of Nujiang, Diqing, Lijiang, Dehong, Baoshan, Kunming and Chuxiong in the Yunnan Province. Another 350,000 Lisu live in Myanmar, Thailand and India. Other user communities are mostly Christians from the Dulong, the Nu and the Bai nationalities in China.

At present, about 200,000 Lisu in China use the Lisu script and about 160,000 in the other countries are literate in it. The Lisu script is widely used in China in education, publishing, the media and religion. Various schools and universities at the national, provincial and prefectural levels have been offering Lisu courses for many years. Globally, the script is also widely used in a variety of Lisu literature.

Structure. There are 40 letters in the Lisu alphabet. These consist of 30 consonants and 10 vowels. Each letter was originally derived from the capital letters of the Latin alphabet. Twenty-five of them look like sans-serif Latin capital letters (all but "Q") in upright positions; the other 15 are derived from sans-serif Latin capital letters rotated 180 degrees.

Although the letters of the Lisu script clearly derived originally from the Latin alphabet, the Lisu script is distinguished from the Latin script. The Latin script is bicameral, with case mappings between uppercase and lowercase letters. The Lisu script is unicameral; it has no casing, and the letters do not change form. Furthermore, typography for the Lisu script is rather sharply distinguished from typography for the Latin script. There is not the same range of font faces as for the Latin script, and Lisu typography is typically monospaced and heavily influenced by the conventions of Chinese typography.

Consonant letters have an inherent [ɑ] vowel unless followed by an explicit vowel letter. Three letters sometimes represent a vowel and sometimes a consonant: U+A4EA LISU LETTER WA, U+A4EC LISU LETTER YA, and U+A4ED LISU LETTER GHA.

Tone Letters. The Lisu script has six tone letters which are placed after the syllable to mark tones. These tone letters are listed in <u>*Table 18-13*</u>, with the tones identified in terms of their pitch contours.

Table 18-13. Lisu Tone Letters

Code	Glyph	Name	Tone
A4F8	.	mya ti	55
A4F9	,	na po	35
A4FA	..	mya cya	44
A4FB	.,	mya bo	33
A4FC	;	mya na	42
A4FD	:	mya jeu	31

Each of the six tone letters represents one simple tone. Although the tone letters clearly derive from Western punctuation marks (full stop, comma, semicolon, and colon), they do not function as punctuation at all. Rather, they are word-forming modifier letters.

The first four tone letters can be used in combination with the last two to represent certain combination tones. Of the various possibilities, only ".;" is still in use; the rest are now rarely seen in China. In monospaced fonts where all letters have the same advance width (for example, one em), it is desirable to fit such a combination of tone letters into the advance width of a simple tone letter.

Other Modifier Letters. Nasalized vowels are denoted by a nasalization mark following the vowel. This word-forming character is not encoded separately in the Lisu script, but is represented by U+02BC MODIFIER LETTER APOSTROPHE, which has the requisite shape and properties (General_Category = Lm) and is used in similar contexts.

A glide based on the vowel *A*, pronounced as [ɑ] without an initial glottal stop (and normally bearing a 31 low falling pitch), is written after a verbal form to mark various aspects. This word-forming modifier letters is represented by U+02CD MODIFIER LETTER LOW MACRON. In a Lisu font, this modifier letter should be rendered on the baseline, to harmonize with the position of the tone letters.

Digits and Separators. There are no unique Lisu digits. The Lisu use European digits for counting. The thousands separator and the decimal point are represented with U+002C COMMA and U+002E FULL STOP, respectively. To separate chapter and verse numbers, U+003A COLON and U+003B SEMICOLON are used. These can be readily distinguished from the similar-appearing tone letters by their numerical context.

Punctuation. U+A4FE "-," LISU PUNCTUATION COMMA and U+A4FF "=" LISU PUNCTUATION FULL STOP are punctuation marks used respectively to denote a lesser and a greater degree of finality. These characters are similar in appearance to sequences of Latin punctuation marks, but are not unified with them.

Over time various other punctuation marks from European or Chinese traditions have been adopted into Lisu orthography. *Table 18-14* lists all known adopted punctuation, along with the respective contexts of use.

Table 18-14. Punctuation Adopted in Lisu Orthography

Code	Glyph	Name	Context
002D	-	hyphen-minus	syllable separation in names
003F	?	question mark	questions
0021	!	exclamation mark	exclamations
0022	"	quotation mark	quotations
0028/0029	( )	parentheses	parenthetical notes
300A/300B	《 》	double angle brackets	book titles
2026	...	ellipsis	omission of words (always doubled in Chinese usage)

U+2010 HYPHEN may be preferred to U+002D HYPHEN-MINUS for the dash used to separate syllables in names, as its semantics are less ambiguous than U+002D.

The use of the U+003F "?" QUESTION MARK replaced the older Lisu tradition of using a tone letter combination to represent the question prosody, followed by a Lisu full stop: "..:="

Line Breaking. A line break is not allowed within an orthographic syllable in Lisu. A line break is also prohibited before a punctuation mark, even if it is preceded by a space. In general there is no hyphenation of words across line breaks, except for proper nouns, where a break is allowed after the hyphen used as a syllable separator.

Word Separation. The Lisu script separates syllables using a space or, for proper names, a hyphen. In the case of polysyllabic words, it can be ambiguous as to which syllables join together to form a word. Thus for most text processing at the character level, a syllable (starting after a space or punctuation and ending before another space or punctuation) is treated as a word except for proper names—where the occurrence of a hyphen holds the word together.

18.10 Miao

18.10.1 Miao: U+16F00–U+16F9F

The Miao script, also called Lao Miaowen ("Old Miao Script") in Chinese, was created in 1904 by Samuel Pollard and others, to write the Northeast Yunnan Miao language of southern China. The script has also been referred to as the Pollard script, but that usage is no longer preferred. The Miao script was created by an adaptation of Latin letter variants, English shorthand characters, Miao pictographs, and Cree syllable forms. (See *Section 20.2, Canadian Aboriginal Syllabics*.) Today, the script is used to write various Miao dialects, as well as languages of the Yi and Lisu nationalities in southern China.

The script was reformed in the 1950s by Yang Rongxin and others, and was later adopted as the "Normalized" writing system of Kunming City and Chuxiong Prefecture. The main difference between the pre-reformed and the reformed orthographies is in how they mark tones. Both orthographies can be correctly represented using the Miao characters encoded in the Unicode Standard.

Implementation Guidelines. Extensive guidelines for the implementation of the Miao script can be found in Unicode Technical Note #56, Representing Miao in Unicode. That document provides information on the encoding order of syllables, on rendering, and on glyph variants. (Unicode Technical Notes do not have normative status for the Unicode Standard.)

Encoding Principles. The script is written left to right. The basic syllabic structure contains an initial consonant or consonant cluster and a final. The final consists of either a vowel or vowel cluster, an optional final nasal, plus a tone mark. The initial consonant may be preceded by U+16F50 MIAO LETTER NASALIZATION, and can be followed by combining marks for voicing (U+16F52 MIAO SIGN REFORMED VOICING) or aspiration (U+16F51 MIAO SIGN ASPIRATION and U+16F53 MIAO SIGN REFORMED ASPIRATION).

The Gan Yi variety of Miao has an additional combining mark, U+16F4F MIAO SIGN CONSONANT MODIFIER BAR. That mark is only applied to two consonants, U+16F0E MIAO LETTER TTA or U+16F10 MIAO LETTER NA, indicating a distinct place of articulation. The mark follows the consonant in logical order, as for all combining marks, but is rendered with a small vertical bar at the lower left-hand side of the modified consonant.

Tone Marks. In the Chuxiong reformed orthography, vowels and final nasals appear on the baseline. If no explicit tone mark is present, this indicates the default tone 3. An additional tone mark, encoded in the range U+16F93..U+16F99, may follow the

vowel to indicate other tones. A set of archaic tone marks used in the reformed orthography is encoded in the range U+16F9A..U+16F9F.

In the pre-reformed orthography, such as that used for the language Ahmao (Northern Hmong), the tone marks are represented in a different manner, using one of five shifter characters. These are represented in sequence following the vowel or vowel sequence and indicate where the vowel letter is to be rendered in relation to the consonant. If more than one vowel letter appears before the shifter, all of the vowel glyphs are moved together to the appropriate position.

Rendering of "wart". Several Miao consonants appear in the code charts with a "wart" attached to the glyph, usually on the left-hand side. In the Chuxiong orthography, a dot appears instead of the wart on these consonants. Because the user communities consider the appearance of the wart or dot to be a different way to write the same characters and not a difference of the character's identity, the differences in appearance are a matter of font style.

Ordering. The order of Miao characters in the code charts derives from a reference ordering widely employed in China, based in part on the order of Bopomofo phonetic characters. The expected collation order for Miao strings varies by language and user communities, and requires tailoring. See Unicode Technical Standard #10, "Unicode Collation Algorithm."

Digits. Miao uses European digits.

Punctuation. The Miao script employs a variety of punctuation marks, both from the East Asian typographical tradition and from the Western typographical tradition. There are no script-specific punctuation marks.

18.11 Tangut

18.11.1 Tangut: U+17000–U+187FF

Tangut Supplement: U+18D00–U+18D7F

Tangut, also known as Xixia, is a large, historic siniform ideographic script used to write the Tangut language, a Tibeto-Burman language spoken from about the 11th century CE until the 16th century in the area of present-day northwestern China. The Tangut script was created under the first emperor of Western Xia about 1036 CE. After the fall of the Western Xia to the Mongols, the script continued to be used during the Yuan and Ming dynasties, but it had become obsolete by the end of Ming dynasty. Tangut was re-discovered in the late 19th century, and has been largely deciphered, thanks to the ground-breaking work done in the early 20th century by N. A. Nevskij. Tangut is found in thousands of official, private, and religious texts, including books and sutras, inscriptions, and manuscripts. Today the study of Tangut is a separate discipline, with scholars in China, Japan, Russia, and other countries publishing works on Tangut language and culture.

Structure. Tangut characters superficially resemble Chinese ideographs; however, the script is unique and unrelated to Chinese ideographs. Tangut was originally written top to bottom, with columns laid out right to left, in the same manner as Chinese was traditionally written. In current practice, the script is written horizontally left to right. Most Tangut characters are made up of 8 to 15 strokes. The script has no combining characters.

Encoding Principles. The repertoire of Tangut characters is intended to cover all Tangut characters used as head entries or index entries in the major works of modern Tangut lexicography and scholarship. A number of principles have been adopted to handle variant glyph shapes, because Tangut characters are often written with different glyph shapes in the primary sources. When character variants are not used contrastively in a single source reference, they are unified as a single character, typically using the glyph found in Li Fanwen 2008. However, if a single source includes two or more variants as separate head or index entries, then the variants are encoded as separate characters. In cases where two characters with the same shape are cataloged separately in a single source, but have different pronunciations or meanings, only one character is encoded. Also, a few erroneous or "ghost" characters in modern dictionaries are separately encoded.

The Tangut Supplement block contains additional Tangut ideographs that did not fit within the initial allocation range for the Tangut block. In some cases, these additional ideographs are disunifications resulting from scholarly analysis of some components that have very closely-related graphical appearances.

Character Names. The names for the Tangut characters are algorithmically derived by prefixing the code point with the string "TANGUT IDEOGRAPH-". Hence the name for U+17000 is TANGUT IDEOGRAPH-17000.

Punctuation. Contemporary sources use U+16FE0 TANGUT ITERATION MARK, located in the Ideographic Symbols and Punctuation block. There are no other script-specific punctuation marks.

Sources. The Unicode Character Database contains a source data file for Tangut called TangutSources.txt. This data file contains normative information on the source references for each Tangut character. TangutSources.txt also contains the informative radical-stroke values for each character. The data in TangutSources.txt shares the same format as the Unihan data files in the UCD. The Tangut code chart also indicates the source reference and the radical-stroke value for each character.

Sorting. No universally accepted or standard character sort order exists for Tangut. All extant Tangut dictionaries dating to the Western Xia period (1038-1227) base their ordering on phonetic principles, which do not help in locating specific characters. Almost all modern Tangut dictionaries and glossaries order characters by radical and stroke count. However, the radical/stroke indices in modern handbooks all differ from one another. The radical system adopted in the Tangut block is based on that of Han Xiaomang 2004, with some modifications. In the Tangut block, signs are grouped by radical, and radicals are ordered by stroke count and stroke order. Within each radical, signs are ordered by stroke count and stroke order.

Stroke Order. Because current day Tangut dictionaries do not provide information on how Tangut characters should be written or on their stroke count, modern scholars have reconstructed stroke count and stroke order based on the analogy to Chinese characters. The stroke order used by scholars may not reflect the actual stroke order used by Tangut scribes.

18.11.2 Tangut Components: U+18800–U+18AFF

Tangut characters are composed of structural elements called components. The components and stroke order are used by scholars to index Tangut ideographs in modern dictionaries and glossaries. The components are also used to describe and analyze Tangut ideographs.

Because there is no single standard set of components, different scholars have devised their own systems. The Tangut Components block represents a unification of seventeen Chinese, Japanese, Russian, and English language dictionaries of Tangut and other publications. All components used in important recent Tangut dictionaries are included, as well as an additional 24 components required for describing Tangut ideographs. The components can be used in Ideographic Description Sequences (IDS) to describe Tangut ideographs.

Repertoire. A total of 755 components are encoded. Of these, 505 components function as radicals under which the Tangut ideographs are ordered. Some sources use single strokes to describe or to index characters. In some cases, these single strokes are encoded as components (U+18900..U+18909), but other single strokes may be represented using the corresponding character from the CJK Strokes block instead.

Names. The characters in the Tangut Components block are named sequentially by prefixing the string "TANGUT COMPONENT-" to a three digit numerical sequence code. Hence, the names range from TANGUT COMPONENT-001 through TANGUT COMPONENT-755.

Order. The Tangut components are ordered by stroke count and stroke order.

Radical-Stroke Values. The Unicode Character Database contains the Tangut radical-stroke values for each character in the data file TangutSources.txt. This data is informative, and is in the same format as Unihan. The Tangut code chart also indicates the source reference and the radical-stroke value for each character.

18.12 Khitan Small Script

18.12.1 Khitan Small Script: U+18B00–U+18CFF

Khitan Small Script was one of two scripts used by the Khitan people of Northern China to write the Khitan language during the Liao dynasty (907–1125 CE), the Qara Khitai empire (or Western Liao dynasty, 1124–1218), and the Jin dynasty (1115–1234). The other script is known as Khitan Large Script. Both scripts are only partially deciphered today but were used over the same time period, in the same geographical area, and for the same functions.

Khitan Small Script was created about 925 by Yelü Diela, and its creation is said to have been inspired by the Uyghur script, although there appear to be few similarities between the two scripts. The main source of texts in Khitan Small Script are funerary epitaphs engraved on stone tablets and buried with members of Khitan royalty and aristocracy. The script also appears on walls and monuments, as well as on bronze mirrors, tallies, non-circulation coins, and a single jade cup.

Structure. The Khitan Small Script contains logograms and phonograms written in vertical columns, running right to left, similar to how Chinese is traditionally written. The logograms generally appear on their own, and phonograms typically combine into clusters of two to eight characters to represent an individual word.

A small number of frequently occurring logograms represent numbers, calendrical terms, kinship terms, and so on. Some of these may appear with dotted and undotted forms. The dotted forms are thought to indicate masculine gender, while the undotted forms indicate feminine gender or are gender-neutral.

Most Khitan words are written phonetically with characters that represent consonants, vowels, diphthongs or syllables. The phonetic values of many of the phonograms have been reconstructed, but many values are still unknown. A few characters seem to act both as logograms and phonograms.

Character Names. The Khitan Small Script characters are named sequentially by prefixing "KHITAN SMALL SCRIPT CHARACTER-" to the code point, with the exception of one format control character, U+16FE4 KHITAN SMALL SCRIPT FILLER. The filler character is located in the Ideographic Symbols and Punctuation block.

Phonogram Clusters. Phonograms may occur in isolation, but typically, two or more phonograms combine into a cluster representing a single word of one or more syllables. Within the cluster, the characters are ordered from left to right and then from top to bottom. Less often, a phonogram starts with a single centered character at the top. Some logograms may take a grammatical suffix and therefore appear as the first character in a phonogram cluster.

There are two cluster patterns in Khitan Small Script. The prevalent pattern, Type A, starts with two side-by-side adjacent Khitan Small Script characters, and ends with either a single centered character or two additional side-by-side adjacent characters. The alternate pattern, Type B, occurs occasionally. It starts with a single, centered Khitan Small Script character at the top, usually followed by two, sometimes three, and very rarely more than three characters, as shown in *Figure 18-17*. The two patterns seem to be a stylistic choice, rather than a semantic distinction.

Figure 18-17. Cluster Patterns in Khitan Small Script

Cluster Type A

2	3	4	5	6	7	8
①②	①② ③	①② ③④	①② ③④ ⑤	①② ③④ ⑤⑥	①② ③④ ⑤⑥ ⑦	①② ③④ ⑤⑥ ⑦⑧

Cluster Type B

2	3	4
① ②	① ②③	① ②③ ④

The original Khitan Small Script texts show a narrow gap between clusters, between clusters and standalone characters, and often between adjacent standalone characters. Modern scholarly transcriptions of texts generally show a clear gap between standalone characters and sequences of characters. To indicate the gap, U+0020 SPACE should be used.

Clusters of Type A are predominant. A rendering system should lay out clusters of this type automatically, by default. To indicate clusters of Type B, the format character U+16FE4 KHITAN SMALL SCRIPT FILLER is used, placed directly after the first character.

Additional rendering support is required to lay out Khitan Small Script in the various attested orientations: in clusters within vertical text, in clusters within left-to-right horizontal text, or simply character-by-character in a horizontal, linear format.

Iteration Mark. Khitan Small Script contains an iteration mark, U+18B00 KHITAN SMALL SCRIPT CHARACTER-18B00. This mark indicates that the preceding cluster is repeated in reading.

Obscured or Missing Characters. Occasionally a Khitan Small Script character may be obscured or missing in source materials, often as a result of damage to inscriptions. In such cases, U+18CFF KHITAN SMALL SCRIPT CHARACTER-18CFF can be used to represent the obscured or missing character. The representative glyph for U+18CFF is a white square box, but it may also be shown with dashed or dotted edges. This symbolic indicator of a missing character participates in Khitan Small Script cluster rendering behavior, and so the aspect and/or size of the box may vary, depending on how the clusters are rendered in context.

Chapter 19

Africa

Ethiopic and Tifinagh are scripts with long histories. Although their roots can be traced back to the original Semitic and North African writing systems, they would not be classified as Middle Eastern scripts today.

The remaining scripts in this chapter have been developed relatively recently. Some of them show roots in Latin and other letterforms. They are all original creative contributions intended specifically to serve the linguistic communities that use them.

Osmanya is an alphabetic script developed in the early 20th century to write the Somali language. N'Ko is a right-to-left alphabetic script devised in 1949 as a writing system for Manden languages in West Africa. Vai is a syllabic script used for the Vai language in Liberia and Sierra Leone; it was developed in the 1830s, but the standard syllabary was published in 1962. Bamum is a syllabary developed between 1896 and 1910, used for writing the Bamum language in western Cameroon. Modern Bassa Vah is an alphabetic script developed early in the 20th century. Mende Kikakui is a right-to-left script used for writing Mende. It was also created in the early 20th century.

Adlam is an alphabetic script used to write Fulani and other African languages. The Fulani are a widespread ethnic group in Africa, and the Fulani language is spoken by more than 40 million people. The script was developed in the late 1980s, and was subsequently widely adopted among Fulani communities, where it is taught in schools.

The Medefaidrin script is used to write the liturgical language Medefaidrin by members of an indigenous Christian church in Nigeria. According to community tradition, the language was revealed to one of the founders of the community in 1927 by divine inspiration. It is presently used for Sunday school lessons and prayers or meditation.

Garay is a right-to-left alphabetic script used to write Wolof in Senegal. It was created by Assane Faye and first published in 1961. It has a small user community in Senegal. The script is designed to be easy to learn and use by people familiar with the Arabic script.

19.1 Ethiopic

19.1.1 Ethiopic: U+1200–U+137F

The Ethiopic syllabary originally evolved for writing the Semitic language Ge'ez. Indeed, the English noun "Ethiopic" simply means "the Ge'ez language." Ge'ez itself is now limited to liturgical usage, but its script has been adopted for modern use in writing several languages of central east Africa, including Amharic, Tigre, and Oromo.

Basic and Extended Ethiopic. The Ethiopic characters encoded here include the basic set that has become established in common usage for writing major languages. As with other productive scripts, the basic Ethiopic forms are sometimes modified to produce an extended range of characters for writing additional languages.

Encoding Principles. The syllables of the Ethiopic script are traditionally presented as a two-dimensional matrix of consonant-vowel combinations. The encoding follows this structure; in particular, the codespace range U+1200..U+1357 is interpreted as a matrix of 43 consonants crossed with 8 vowels, making 344 conceptual syllables. Most of these consonant-vowel syllables are represented by characters in the script, but some of them happen to be unused, accounting for the blank cells in the matrix.

Variant Glyph Forms. A given Ethiopic syllable may be represented by different glyph forms, analogous to the glyph variants of Latin lowercase "a" or "g", which do not coexist in the same font. Thus the particular glyph shown in the code chart for each position in the matrix is merely one representation of that conceptual syllable, and the glyph itself is not the object that is encoded.

Labialized Subseries. A few Ethiopic consonants have labialized ("W") forms that are traditionally allotted their own consonant series in the syllable matrix, although only a subset of the possible vowel forms are realized. Each of these derivative series is encoded immediately after the corresponding main consonant series. Because the standard vowel series includes both "AA" and "WAA", two different cells of the syllable matrix might represent the "consonant + W + AA" syllable. For example:

U+1257 = QH + WAA: potential but unused version of QHWAA

U+125B = QHW + AA: ETHIOPIC SYLLABLE QHWAA

In these cases, where the two conceptual syllables are equivalent, the entry in the labialized subseries is encoded and not the "consonant + WAA" entry in the main syllable series. The six specific cases are enumerated in . In three of these cases, the -WAA position in the syllable matrix has been reanalyzed and used for encoding a syllable in -OA for extended Ethiopic.

Table 19-1. Labialized Forms in Ethiopic -WAA

-WAA Form	Encoded as	Not Used	Contrast
QWAA	U+124B ቋ	1247	U+1247 ቇ QOA
QHWAA	U+125B ቛ	1257	
XWAA	U+128B ኋ	1287	U+1287 ኇ XOA
KWAA	U+12B3 ኳ	12AF	U+12AF ኯ KOA
KXWAA	U+12C3 ዃ	12BF	
GWAA	U+1313 ጓ	130F	

Also, *within* the labialized subseries, the sixth vowel ("-E") forms are sometimes considered to be second vowel ("-U") forms. For example:

U+1249 = QW + U: unused version of QWE

U+124D = QW + E: ETHIOPIC SYLLABLE QWE

In these cases, where the two syllables are nearly equivalent, the "-E" entry is encoded and not the "-U" entry. The six specific cases are enumerated in .

Table 19-2. Labialized Forms in Ethiopic -WE

"-WE" Form	Encoded as	Not Used
QWE	U+124D ቍ	1249
QHWE	U+125D ቝ	1259
XWE	U+128D ኍ	1289
KWE	U+12B5 ኵ	12B1
KXWE	U+12C5 ዅ	12C1
GWE	U+1315 ጕ	1311

Keyboard Input. Because the Ethiopic script includes more than 300 characters, the units of keyboard input must constitute some smaller set of entities, typically 43+8 codes interpreted as the coordinates of the syllable matrix. Because these keyboard input codes are expected to be transient entities that are resolved into syllabic characters before they enter stored text, keyboard input codes are not specified in this standard.

Syllable Names. The Ethiopic script often has multiple syllables corresponding to the same Latin letter, making it difficult to assign unique Latin names. Therefore the names list makes use of certain devices (such as doubling a Latin letter in the name) merely to create uniqueness; this device has no relation to the phonetics of these syllables in any particular language.

Encoding Order and Sorting. The order of the consonants in the encoding is based on the traditional alphabetical order. It may differ from the sort order used for one or another language, if only because in many languages various pairs or triplets of syllables are treated as equivalent in the first sorting pass. For example, an Amharic dictionary may start out with a section headed by *three* H-like syllables:

U+1200 ETHIOPIC SYLLABLE HA

U+1210 ETHIOPIC SYLLABLE HHA

U+1280 ETHIOPIC SYLLABLE XA

Thus the encoding order cannot and does not implement a collation procedure for any particular language using this script.

Diacritical Marks. The Ethiopic script generally makes no use of diacritical marks, but they are sometimes employed for scholarly or didactic purposes. In particular, U+135F ETHIOPIC COMBINING GEMINATION MARK and U+030E COMBINING DOUBLE VERTICAL LINE ABOVE are sometimes used to indicate emphasis or gemination (consonant doubling).

Numbers. Ethiopic digit glyphs are derived from the Greek alphabet, possibly borrowed from Coptic letterforms. In modern use, European digits are often used. The Ethiopic number system does not use a zero, nor is it based on digital-positional notation. A number is denoted as a sequence of powers of 100, each preceded by a coefficient (2 through 99). In each term of the series, the power 100^n is indicated by n HUNDRED characters (merged to a digraph when n = 2). The coefficient is indicated by a *tens* digit and a *ones* digit, either of which is absent if its value is zero.

For example, the number 2,345 is represented by

2,345 = (20 + 3)*100^1 + (40 + 5)*100^0

= 20 3 100 40 5

= TWENTY THREE HUNDRED FORTY FIVE

= 1373 136B 137B 1375 136D ፳፫፻፵፭

A language using the Ethiopic script may have a *word* for "thousand," such as Amharic "SHI" (U+123A), and a quantity such as 2,345 may also be written as it is spoken in that language, which in the case of Amharic happens to parallel English:

2,345 = TWO thousand THREE HUNDRED FORTY FIVE

= 136A 123A 136B 137B 1375 136D ፪ሺ፫፻፵፭

In Ge'ez language manuscripts the conjunction "**ω**" is frequently used to write numbers as they would be spoken.

For example, the number 2,345 would then be written in a Ge'ez language document as

2,345 = TWENTY and THREE HUNDRED with FORTY and FIVE

= 136A 12C8 136B 137B 12C8 1375 12C8 136D ፳ወ፫፻፸ወ፵ወ፭

Word Separators. The traditional word separator is U+1361 ETHIOPIC WORDSPACE (:). In modern usage, a plain white wordspace (U+0020 SPACE) is becoming common.

Section Mark. One or more *section marks* are typically used on a separate line to mark the separation of sections. Commonly, an odd number is used and they are separated by spaces.

19.1.2 Ethiopic Extensions

The Ethiopic script is used for a large number of languages and dialects in Ethiopia and in some instances has been extended significantly beyond the set of characters used for major languages such as Amharic and Tigre. There are four blocks of extensions to the Ethiopic script: Ethiopic Supplement U+1380..U+139F, Ethiopic Extended U+2D80..U+2DDF, Ethiopic Extended-A U+AB00..U+AB2F, and Ethiopic Extended-B U+1E7E0..U+1E7FF. Those extensions cover such languages as Me'en, Blin, and the Gurage languages, which use many additional characters. The Ethiopic Extended-A block, in particular, includes characters for the Gamo-Gofa-Dawro, Basketo, and Gumuz languages. Several other characters for Ethiopic script extensions can be found in the main Ethiopic script block in the range U+1200..U+137F, including combining diacritical marks used for Basketo.

The Ethiopic Extended-B block contains characters for the modern Gurage orthography, which covers the Inor, Mesqan, Sebatbeit, and Soddo languages. Additional characters for this orthography can be found in the Ethiopic Supplement and Ethiopic Extended blocks. Some of the character names in these blocks include the word "SEBATBEIT" because they were originally encoded for the older Sebatbeit orthography. The modern Gurage orthography uses some of these characters for all Gurage languages, including Sebatbeit.

The Ethiopic Supplement block also contains a set of tonal marks. They are used in multiline scored layout. Like other musical (an)notational systems of this type, these tonal marks require a higher-level protocol to enable proper rendering.

19.2 Osmanya

19.2.1 Osmanya: U+10480–U+104AF

The Osmanya script, which in Somali is called 𐒄𐒈𐒇 𐒈𐒓𐒈𐒁𐒍𐒘𐒋 *far Soomaali* "Somali writing" or 𐒋𐒘𐒈𐒁𐒒𐒈𐒐𐒈 *Cismaanya*, was devised in 1920–1922 by 𐒋𐒘𐒈𐒁𐒒 𐒕𐒓𐒈𐒕𐒓 𐒆𐒚𐒊𐒇𐒈𐒁𐒘𐒆 (Cismaan Yuusuf Keenadiid) to represent the Somali language. It replaced an attempt by Sheikh Uweys of the Confraternity Qadiriyyah (died 1909) to devise an Arabic-based orthography for Somali. It has, in turn, been replaced by the Latin orthography of Muuse Xaaji Ismaaciil Galaal (1914–1980). In 1961, both the Latin and the Osmanya scripts were adopted for use in Somalia, but in 1969 there was a coup, with one of its stated aims being the resolution of the debate over the country's writing system. A Latin orthography was finally adopted in 1973. Gregersen (1977) states that some 20,000 or more people use Osmanya in private correspondence and bookkeeping, and that several books and a biweekly journal *Horseed* ("*Vanguard*") were published in cyclostyled format.

Structure. Osmanya is an alphabetic script, read from left to right in horizontal lines running from top to bottom. It has 22 consonants and 8 vowels. Unique long vowels are written for U+1049B 𐒛 osmanya letter aa, U+1049C 𐒜 osmanya letter ee, and U+1049D 𐒝 osmanya letter oo; long *uu* and *ii* are written with the consonants U+10493 𐒓 osmanya letter waw and U+10495 𐒕 osmanya letter ya, respectively.

Ordering. Alphabetical ordering is based on the order of the Arabic alphabet, as specified by Osman Abdihalim Yuusuf Osman Keenadiid. This ordering is similar to the ordering given in Diringer (1996).

Character Names and Glyphs. The character names used in the Unicode Standard are as given by Osman. The glyphs shown in the code charts are taken from *Afkeenna iyo fartysa* ("Our language and its handwriting") 1971.

19.3 Tifinagh

19.3.1 Tifinagh: U+2D30–U+2D7F

The Tifinagh script is used by approximately 20 million people who speak varieties of languages commonly called Berber or Amazigh. The three main varieties in Morocco are known as Tarifite, Tamazighe, and Tachelhite. In Morocco, more than 40% of the population speaks Berber. The Berber language, written in the Tifinagh script, is currently taught to approximately 300,000 pupils in 10,000 schools—mostly primary schools—in Morocco. Three Moroccan universities offer Berber courses in the Tifinagh script leading to a Master's degree.

Tifinagh is an alphabetic writing system. It uses spaces to separate words and makes use of Western punctuation.

History. The earliest variety of the Berber alphabet is Libyan. Two forms exist: a Western form and an Eastern form. The Western variety was used along the Mediterranean coast from Kabylia to Morocco and most probably to the Canary Islands. The Eastern variety, Old Tifinagh, is also called Libyan-Berber or Old Tuareg. It contains signs not found in the Libyan variety and was used to transcribe Old Tuareg. The word *tifinagh* is a feminine plural noun whose singular would be *tafniqt*; it means "the Phoenician (letters)."

Neo-Tifinagh refers to the writing systems that were developed to represent the Maghreb Berber dialects. A number of variants of Neo-Tifinagh exist, the first of which was proposed in the 1960s by the Académie Berbère. That variant has spread in Morocco and Algeria, especially in Kabylia. Other Neo-Tifinagh systems are nearly identical to the Académie Berbère system. The encoding in the Tifinagh block is based on the Neo-Tifinagh systems.

Source Standards. The encoding consists of four Tifinagh character subsets: the basic set of the Institut Royal de la Culture Amazighe (IRCAM), the extended IRCAM set, other Neo-Tifinagh letters in use, and modern Tuareg letters. The first subset represents the set of characters chosen by IRCAM to unify the orthography of the different Moroccan modern-day Berber dialects while using the historical Tifinagh script.

Ordering. The letters are arranged according to the order specified by IRCAM. Other Neo-Tifinagh and Tuareg letters are interspersed according to their pronunciation.

Directionality. Historically, Berber texts did not have a fixed direction. Early inscriptions were written horizontally from left to right, from right to left, vertically (bottom to top, top to bottom); boustrophedon directionality was also known. Modern-day Berber script is most frequently written in horizontal lines from left to right; there-

fore the bidirectional class for Tifinagh letters is specified as strong left-to-right. Displaying Berber texts in other directions can be accomplished by the use of directional overrides or by the use of higher-level protocols.

Diacritical Marks. Modern Tifinagh variants tend to use combining diacritical marks to complement the Tifinagh block. The Hawad notation, for example, uses diacritical marks from the Combining Diacritical Marks block (U+0300–U+036F). These marks are used to represent vowels and foreign consonants. In this notation, <U+2D35, U+0307> represents "a", <U+2D49, U+0309> represents a long "i" /iː/, and <U+2D31, U+0302> represents a "p". Some long vowels are represented using two diacritical marks above. A long "e" /eː/ is thus written <U+2D49, U+0307, U+0304>. These marks are displayed side by side above their base letter in the order in which they are encoded, instead of being stacked.

Yal and Yan. While the neo-Tifinagh glyph for U+2D4D TIFINAGH LETTER YAL in Morocco is typically rendered with two bars linked by a small slanted stroke ⵍ, traditional texts from all areas, and modern-day materials from areas outside Morocco often represent *yal* with two vertical strokes ‖. However, the two vertical bar shape can cause visual ambiguity in words with consonant clusters, because *yal* may be mistaken for two instances of U+2D4F TIFINAGH LETTER YAN, whose glyph is a single vertical stroke ⵏ. Individual font designers, local traditions, and national preferences employ various means to prevent confusion, including varying the spacing between the bars, and slanting or lowering the bars. *Figure 19-1* shows examples that illustrate contextual shaping by slanting the bars of *yal* and *yan*.

Figure 19-1. Tifinagh Contextual Shaping
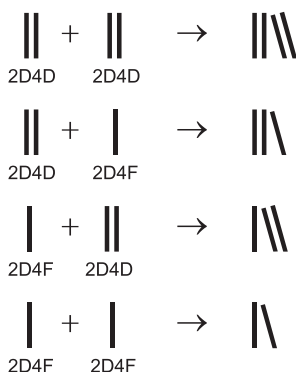
Bi-Consonants. Bi-consonants are additional letterforms used in the Tifinagh script, particularly for Tuareg, to represent a consonant cluster—a sequence of two consonants without an intervening vowel. These bi-consonants, sometimes also referred to as bigraphs, are not directly encoded as single characters in the Unicode Standard. Instead, they are represented as a sequence of the two consonant letters, separated

either by U+200D ᴢᴇʀᴏ ᴡɪᴅᴛʜ ᴊᴏɪɴᴇʀ or by U+2D7F ᴛɪꜰɪɴᴀɢʜ ᴄᴏɴꜱᴏɴᴀɴᴛ ᴊᴏɪɴᴇʀ.

When a bi-consonant is considered obligatory in text, it is represented by the two consonant letters, with U+2D7F ᴛɪꜰɪɴᴀɢʜ ᴄᴏɴꜱᴏɴᴀɴᴛ ᴊᴏɪɴᴇʀ inserted between them. This use of U+2D7F is comparable in function to the use of U+0652 ᴀʀᴀʙɪᴄ ꜱᴜᴋᴜɴ to indicate the absence of a vowel after a consonant, when Tuareg is written in the Arabic script. However, instead of appearing as a visible mark in the text, U+2D7F ᴛɪꜰɪɴᴀɢʜ ᴄᴏɴꜱᴏɴᴀɴᴛ ᴊᴏɪɴᴇʀ indicates the presence of a bi-consonant, which should then be rendered with a preformed glyph for the sequence. Examples of common Tifinagh bi-consonants and their representation are shown in *Figure 19-2*.

Figure 19-2. Tifinagh Consonant Joiner and Bi-consonants
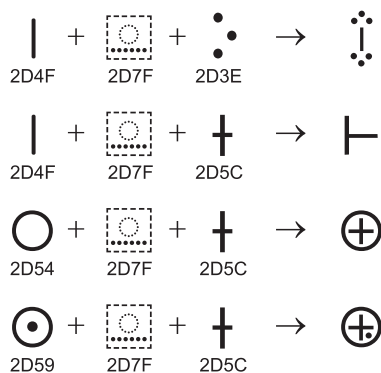
If a rendering system cannot display obligatory bi-consonants with the correct, fully-formed bi-consonant glyphs, a fallback rendering should be used which displays the ᴛɪꜰɪɴᴀɢʜ ᴄᴏɴꜱᴏɴᴀɴᴛ ᴊᴏɪɴᴇʀ visibly, so that the correct textual distinctions are maintained, even if they cannot be properly displayed.

When a bi-consonant is considered merely an optional, ligated form of two consonant letters, the bi-consonant can be represented by the two consonant letters, with U+200D ᴢᴇʀᴏ ᴡɪᴅᴛʜ ᴊᴏɪɴᴇʀ inserted between them, as a hint that the ligated form is preferred. If a rendering system cannot display the optional, ligated form, the fallback display should simply be the sequence of consonants, with no visible display of the ZWJ.

Bi-consonants often have regional glyph variants, so fonts may need to be designed differently for different regional uses of the Tifinagh script.

19.4 N'Ko

19.4.1 N'Ko: U+07C0–U+07FF

N'Ko is a literary dialect used by the Manden (or Manding) people, who live primarily in West Africa. The script was devised by Solomana Kante in 1949 as a writing system for the Manden languages. The Manden language group is known as *Mandenkan*, where the suffix -*kan* means "language of." In addition to the substantial number of Mandens, some non-Mandens speak *Mandenkan* as a second language. There are an estimated 20 million Mandenkan speakers.

The major dialects of the Manden language are Bamanan, Jula, Maninka, and Mandinka. There are a number of other related dialects. When Mandens from different subgroups talk to each other, it is common practice for them to switch—consciously or subconsciously—from their own dialect to the conventional, literary dialect commonly known as *Kangbe*, "the clear language," also known as N'Ko. This dialect switching can occur in conversations between the Bamanan of Mali, the Maninka of Guinea, the Jula of the Ivory Coast, and the Mandinka of Gambia or Senegal, for example. Although there are great similarities between their dialects, speakers sometimes find it necessary to switch to *Kangbe* (N'Ko) by using a common word or phrase, similar to the accommodations Danes, Swedes, and Norwegians sometimes make when speaking to one another. For example, the word for "name" in Bamanan is *togo*, while it is *tooh* in Maninka. Speakers of both dialects will write it as ߕߐ , although each may pronounce it differently.

Character Names and Block Name. Although the traditional name of the N'Ko language and script includes an apostrophe, apostrophes are disallowed in Unicode character and block names. Because of this, the formal block name is "NKo" and the script portion of the Unicode character names is "NKO".

Structure. The N'Ko script is written from right to left. It is phonetic in nature (one symbol, one sound). N'Ko has seven vowels, each of which can bear one of seven diacritical marks that modify the tone of the vowel as well as an optional diacritical mark that indicates nasalization. N'Ko has 19 consonants and two "abstract" consonants, U+07E0 NKO LETTER NA WOLOSO and U+07E7 NKO LETTER NYA WOLOSO, which indicate original consonants mutated by a preceding nasal, either word-internally or across word boundaries. Some consonants can bear one of three diacritical marks to transcribe foreign sounds or to transliterate foreign letters.

U+07D2 NKO LETTER N is considered neither a vowel nor a consonant; it indicates a syllabic alveolar or velar nasal. It can bear a diacritical mark, but cannot bear the nasal diacritic. The letter U+07D1 NKO LETTER DAGBASINNA has a special function in N'Ko orthography. The standard spelling rule is that when two successive syllables have the same vowel, the vowel is written only after the second of the two syllables.

For example, ߓߟߏ <ba, la, oo> is pronounced [bolo], but in a foreign syllable to be pronounced [blo], the *dagbasinna* is inserted for ߓ߭ߟߏ <ba, dagbasinna, la, oo> to show that a consonant cluster is intended.

Diacritical Marks. N'Ko diacritical marks are script-specific, despite superficial resemblances to other diacritical marks encoded for more general use. Some N'Ko diacritics have a wider range of glyph representation than the generic marks do, and are typically drawn rather higher and bolder than the generic marks.

Two of the tone diacritics, when applied to consonants, indicate specific sounds from other languages—in particular, Arabic or French language sounds. U+07F3 NKO COMBINING DOUBLE DOT ABOVE is also used as a diacritic to represent sounds from other languages. The combinations used are as shown in *Table 19-3*.

Table 19-3. N'Ko Diacritic Usage

Character	Applied To	Represents
U+07EB NKO COMBINING SHORT HIGH TONE	SA	[s] or Arabic ص SAD
	GBA	[ɣ] or Arabic غ GHAIN
	KA	[q] or Arabic ق QAF
U+07ED NKO COMBINING SHORT RISING TONE	BA	[ɓ]
	TA	[ɫ] or Arabic ط TAH
	JA	[z] or Arabic ز ZAIN
	CHA	[ð] or Arabic ذ THAL and also French [ʒ]
	DA	[ɗ] or Arabic ض DAD
	RA	French [ʀ]
	SA	[ʃ] or Arabic ش SHEEN
	GBA	[g]
	FA	[v]
	KA	[x] or Arabic خ KHAH
	LA	[lˤ]
	MA	[ɱ]
	NYA	[ŋ]
	HA	[ħ] or Arabic ح HAH
	YA	[j]
U+07F3 NKO COMBINING DOUBLE DOT ABOVE	A	[ʕa] or Arabic ع AIN + A
	EE	French [ə]
	U	French [y]
	JA	[ẓ] or Arabic ظ ZAH
	DA	[ḍ]

SA	[θ] or Arabic ث THEH
GBA	[kp]

Table 19-4 shows the use of the tone diacritics when applied to vowels.

Table 19-4. N'Ko Tone Diacritics on Vowels

Character		Tone	Applied To
U+07EB	NKO COMBINING SHORT HIGH TONE	high	short vowel
U+07EC	NKO COMBINING SHORT LOW TONE	low	short vowel
U+07ED	NKO COMBINING SHORT RISING TONE	rising-falling	short vowel
U+07EE	NKO COMBINING LONG DESCENDING TONE	descending	long vowel
U+07EF	NKO COMBINING LONG HIGH TONE	high	long vowel
U+07F0	NKO COMBINING LONG LOW TONE	long low	long vowel
U+07F1	NKO COMBINING LONG RISING TONE	rising	long vowel

When applied to a vowel, U+07F2 NKO COMBINING NASALIZATION MARK indicates the nasalization of that vowel. In the text stream, this mark is applied before any of the tone marks because combining marks below precede combining marks above in canonical order.

Digits. N'Ko uses decimal digits specific to the script. These digits have strong right-to-left directionality. Numbers are stored in text in logical order with most significant digit first; when displayed, numerals are then laid out in right-to-left order, with the most significant digit at the rightmost side, as illustrated for the numeral 144 in *Figure 19-3*. This situation differs from how numerals are handled in Hebrew and Arabic, where numerals are laid out in left-to-right order, even though the overall text direction is right to left.

Ordinal Numbers. Diacritical marks are also used to mark ordinal numbers. The first ordinal is indicated by applying U+07ED NKO COMBINING SHORT RISING TONE (a dot above) to U+07C1 NKO DIGIT ONE. All other ordinal numbers are indicated by applying U+07F2 NKO COMBINING NASALIZATION MARK (an oval dot below) to the last digit in any sequence of digits composing the number. Thus the nasalization mark under the digit two would indicate the ordinal value 2nd, while the nasalization mark under the final digit four in the numeral 144 would indicate the ordinal value 144th, as shown in *Figure 19-3*.

Figure 19-3. Examples of N'Ko Ordinals

߁	1st
߂	2nd
߃	3rd
߁߄߄	144th

Punctuation. N'Ko uses a number of punctuation marks in common with other scripts. U+061F ARABIC QUESTION MARK, U+060C ARABIC COMMA, U+061B ARABIC SEMICOLON, and the paired U+FD3E ORNATE LEFT PARENTHESIS and U+FD3F ORNATE RIGHT PARENTHESIS are used, often with different shapes than are used in Arabic. A script-specific U+07F8 NKO COMMA and U+07F9 NKO EXCLAMATION MARK are encoded. The NKO COMMA differs in shape from the ARABIC COMMA, and the two are sometimes used distinctively in the same N'Ko text.

The character U+07F6 NKO SYMBOL OO DENNEN is used as an addition to phrases to indicate remote future placement of the topic under discussion. The decorative U+07F7 ⵣ NKO SYMBOL GBAKURUNEN represents the three stones that hold a cooking pot over the fire and is used to end major sections of text.

The two tonal apostrophes, U+07F4 NKO HIGH TONE APOSTROPHE and U+07F5 NKO LOW TONE APOSTROPHE, are used to show the elision of a vowel while preserving the tonal information of the syllable. Their glyph representations can vary in height relative to the baseline. N'Ko also uses a set of paired punctuation, U+2E1C LEFT LOW PARAPHRASE BRACKET and U+2E1D RIGHT LOW PARAPHRASE BRACKET, to indicate indirect quotations.

Ordering. The order of N'Ko characters in the code charts reflects the traditional ordering of N'Ko. However, in collation, the three archaic letters U+07E8 NKO LETTER JONA JA, U+07E9 NKO LETTER JONA CHA, and U+07EA NKO LETTER JONA RA should be weighted as variants of U+07D6 NKO LETTER JA, U+07D7 NKO LETTER CHA, and U+07D9 NKO LETTER RA, respectively.

Rendering. N'Ko letters have shaping behavior similar to that of Arabic. Each letter can take one of four possible forms, as shown in *Table 19-5*.

Table 19-5. N'Ko Letter Shaping

Character	X_n	X_r	X_m	X_l
A	｜	l	⊥	J
EE	o	ه	ه	ه
I	Y	Y	Y	У
E	∧	∧	∧	∧
U	ப	ப	⊔	⊔
OO	⊐	⊐	⊐	⊐
O	९	९	٩	९
DAGBASINNA	٤	٤	٤	٤
N	٩	٩	٩	٩
BA	F	Ɛ	Ɛ	Ʃ
PA	٦	٦	٦	٦

TA	ᖯ	ᖯ	ᖯ	ᖯ
JA	ᒣ	ᒣ	ᒣ	ᒣ
CHA	ᆌ	ᆌ	ᆌ	ᆌ
DA	ᘻ	ᘻ	ᘻ	ᘻ
RA	ᛏ	ᛏ	ᛏ	ᛏ
RRA	ᛢ	ᛢ	ᛢ	ᛢ
SA	�口	�口	�口	�口
GBA	ᐁ	ᐁ	ᐁ	ᐁ
FA	ᛒ	ᛒ	ᛒ	ᛒ
KA	ᛙ	ᛙ	ᛙ	ᛙ
LA	ᛕ	ᛕ	ᛕ	ᛕ
NA WOLOSO	ᛁ	ᛁ	ᛁ	ᛁ
MA	ᐃ	ᐃ	ᐃ	ᐃ
NYA	ᚱ	ᚱ	ᚱ	ᚱ
NA	ᚵ	ᚵ	ᚵ	ᚵ
HA	ᚩ	ᚩ	ᚩ	ᚩ
WA	ᚪ	ᚪ	ᚪ	ᚪ
YA	◊	◊	◊	◊
NYA WOLOSO	ᚳ	ᚳ	ᚳ	ᚳ
JONA JA	ᚷ	ᚷ	ᚷ	ᚷ
JONA CHA	ᚸ	ᚸ	ᚸ	ᚸ
JONA RA	ᛏ	ᛏ	ᛏ	ᛏ

A noncursive style of N'Ko writing exists where no joining line is used between the letters in a word. This is a font convention, not a dynamic style like bold or italic, both of which are also valid dynamic styles for N'Ko. Noncursive fonts are mostly used as display fonts for the titles of books and articles. U+07FA ɴᴋᴏ ʟᴀᴊᴀɴʏᴀʟᴀɴ is sometimes used like U+0640 ᴀʀᴀʙɪᴄ ᴛᴀᴛᴡᴇᴇʟ to justify lines, although Latin-style justification where space is increased tends to be more common.

19.5 Vai

19.5.1 Vai: U+A500–U+A63F

The Vai script is used for the Vai language, spoken in coastal areas of western Liberia and eastern Sierra Leone. It was developed in the early 1830s primarily by Mọmọlu Duwalu Bukẹlẹ of Jondu, Liberia, who later stated that the inspiration had come to him in a dream. He may have also been aware of, and influenced by, other scripts including Latin, Arabic, and possibly Cherokee, or he may have phoneticized and regularized an earlier pictographic script. In the years afterward, the Vai built an educational infrastructure that enabled the script to flourish; by the late 1800s European traders reported that most Vai were literate in the script. Although there were standardization efforts in 1899 and again at a 1962 conference at the University of Liberia, nowadays the script is learned informally and there is no means to ensure adherence to a standardized version; most Vai literates know only a subset of the standardized characters. The script is primarily used for correspondence and record-keeping, mainly among merchants and traders. Literacy in Vai coexists with literacy in English and Arabic.

Sources. The primary sources for the Vai characters in Unicode are the 1962 Vai Standard Syllabary, modern primers and texts which use the Standard Syllabary (including a few glyph modifications reflecting modern preferences), the 1911 additions of Momolu Massaquoi, and the characters found in *The Book of Ndole*, the longest surviving text from the early period of Vai script usage.

Basic Structure. Vai is a syllabic script written left to right. The Vai language has seven oral vowels [e i a o u ɔ ɛ], five of which also occur in nasal form [ĩ ã ũ ɔ̃ ɛ̃]. The standard syllabary includes standalone vowel characters for the oral vowels and three of the nasal ones, characters for most of the consonant-vowel combinations formed from each of thirty consonants or consonant clusters, and a character for the final velar nasal consonant [ŋ].

The writing system has a *moraic* structure: the weight (or duration) of a syllable determines the number of characters used to write it (as with Japanese kana). A short syllable is written with any single character in the range U+A500..U+A60B. Long syllables are written with two characters, and involve a long vowel, a diphthong, or a syllable ending with U+A60B VAI SYLLABLE NG. Note that the only closed syllables in Vai—that is, those that end with a consonant—are those ending with VAI SYLLABLE NG. The long vowel is generally written using either an additional standalone vowel to double the vowel sound of the preceding character, or using U+A60C VAI SYLLABLE LENGTHENER, while the diphthong is generally written using an additional standalone vowel. In some cases, the second character for a long vowel or diphthong may be written using characters such as U+A54C VAI SYLLABLE HA or U+A54E VAI SYLLABLE WA instead of standalone vowels.

Historic Syllables. In *The Book of Ndole* more than one character may be used to represent a pronounced syllable; they have been separately encoded.

Logograms. The oldest Vai texts used an additional set of symbols called "logograms," representing complete syllables with an associated meaning or range of meanings; these symbols may be remnants from a precursor pictographic script. At least two of these symbols are still used: U+A618 VAI SYMBOL FAA represents the word meaning "die, kill" and is used alongside a person's date of death (the glyph is said to represent a wilting tree); U+A613 VAI SYMBOL FEENG represents the word meaning "thing."

Digits. In the 1920s ten decimal digits were devised for Vai; these digits were "Vai-style" glyph variants of European digits. They never became popular with Vai people, but are encoded in the standard for historical purposes. Modern literature uses European digits.

Punctuation. Vai makes use of European punctuation, although a small number of script-specific punctuation marks commonly occur. U+A60D VAI COMMA rests on or slightly below the baseline; U+A60E VAI FULL STOP rests on the baseline and can be doubled for use as an exclamation mark. U+A60F VAI QUESTION MARK also rests on the baseline; it is rarely used. Some modern primers prefer these Vai punctuation marks; some prefer the European equivalents. Some Vai writers mark the end of a sentence by using U+A502 VAI SYLLABLE HEE instead of punctuation.

Segmentation. Vai is written without spaces between words. Line breaking opportunities can occur between most characters except that line breaks should not occur before U+A60B VAI SYLLABLE NG used as a syllable final, or before U+A60C VAI SYLLABLE LENGTHENER (which is always a syllable final). Line breaks also should not occur before one of the "h-" characters (U+A502, U+A526, U+A54C, U+A573, U+A597, U+A5BD, U+A5E4) when it is used to extend the vowel of the preceding character (that is, when it is a syllable final), and line breaks should not occur before the punctuation characters U+A60D VAI COMMA, U+A60E VAI FULL STOP, and U+A60F VAI QUESTION MARK.

Ordering. There is no evidence of traditional conventions on ordering apart from the order of listings found in syllabary charts. The syllables in the Vai block are arranged in the order recommended by a panel of Vai script experts. Logograms should be sorted by their phonetic values.

19.6 Bamum

19.6.1 Bamum: U+A6A0–U+A6FF

The Bamum script is used for the Bamum language, spoken primarily in western Cameroon. It was developed between 1896 and 1910, mostly by King Ibrahim Njoya of the Bamum Kingdom. Apparently inspired by a dream and by awareness of other writing, his original idea for the script was to collect and provide approximately 500 logographic symbols (denoting objects and actions) to serve more as a memory aid than as a representation of language.

Using the rebus principle, the script was rapidly simplified through six stages, known as Stage A, Stage B, and so on, into a syllabary known as *A-ka-u-ku*, consisting of 80 syllable characters or letters. These letters are used with two combining diacritics and six punctuation marks. The repertoire in this block covers the *A-ka-u-ku* syllabary, or Phase G form, which remains in modern use.

Structure. Modern Bamum is written from left to right. One interesting feature is that sometimes more letters than necessary are used to write a given syllable. For example, the word *lam* "wedding" is written using the sequence of syllabic characters, *la + a + m*. This feature is known as pleonastic syllable representation.

Diacritical Marks. U+A6F0 BAMUM COMBINING MARK KOQNDON may be applied to any of the 80 letters. It usually functions to glottalize the final vowel of a syllable. U+A6F1 BAMUM COMBINING MARK TUKWENTIS is only known to be used with 13 letters—usually to truncate a full syllable to its final consonant.

Punctuation. U+A6F2 BAMUM NJAEMLI was a character used in the original set of logographic symbols to introduce proper names or to change the meaning of a word. The shape of the glyph for *njaemli* has changed, but the character is still in use. The other punctuation marks correspond in function to the similarly-named punctuation marks used in European typography.

Digits. The last ten letters in the syllabary are also used to represent digits. Historically, the last of these was used for 10, but its meaning was changed to represent zero when decimal-based mathematics was introduced.

19.6.2 Bamum Supplement: U+16800–U+16A3F

The Bamum Supplement block contains archaic characters no longer used in the modern Bamum orthography. These historical characters are analogous in some ways to the medievalist characters encoded for the Latin script. Most Bamum writers do not use them, but they are used by specialist linguists and historians.

The main source for the repertoire of Bamum extensions is an analysis in Dugast and Jeffreys 1950. The Bamum script was developed in six phases, labeled with letters. Phase A is the earliest form of the script. Phase G is the modern script encoded in the main Bamum block. The Bamum Supplement block covers distinct characters from the earlier phases which are no longer part of the modern Bamum script.

The character names in this block include a reference to the last phase in which they appear. So, for example, U+16867 BAMUM LETTER PHASE-B PIT was last used during Phase B, while U+168EE BAMUM LETTER PHASE-C PIN continued in use and is attested through Phase C.

Traditional Bamum texts using these historical characters do not use punctuation or digits. Numerical values for digits are written out as words instead.

19.7 Bassa Vah

19.7.1 Bassa Vah: U+16AD0–U+16AFF

The Bassa Vah script is used for the tonal Bassa language of Liberia, which is distinct from the Basa language of Nigeria and the Basaa (sometimes also spelled Bassa) language of Cameroon. Its modern usage and perhaps form are due primarily to Dr. Thomas Flo Lewis in the early 1900s. According to Bassa tradition, an earlier ideographic script was simplified around 1800 by a man named Dirah, and then remained in use primarily among Bassa brought to the Americas as slaves (as was Dirah). While studying abroad in the United States, Lewis learned a version of that script from Dirah's son Jenni and possibly others of Bassa origin in the Americas, and may have made further improvements. The script may also have been influenced by Vai. Lewis actively published about the script; he also arranged for the production of a typesetting machine and primers for Vah, and on his return to Liberia promoted education in the script.

Structure. Modern Bassa Vah is a simple alphabetic script, written from left to right, consisting of 23 consonants, 7 vowels, and 5 tone marks. Except for discussions about the alphabet itself, the vowel letters are always written with tone marks; these marks are placed in a central open area of each vowel glyph. The tone marks are encoded as combining characters.

Punctuation and Digits. Bassa Vah uses a script-specific full stop resembling a plus sign, as well as the European comma, full stop, and quotation marks. It also uses the European digits 0–9.

19.8 Mende Kikakui

19.8.1 Mende Kikakui: U+1E800–U+1E8DF

The Mende Kikakui script is used for the Mende language of Sierra Leone. It is named Kikakui after the sound of its first three characters. Kikakui was popular for correspondence and record-keeping. However, during the 1940s it was largely supplanted by a Latin-based orthography promoted by the British-established Protectorate Literacy Bureau.

An early version of 42 characters was created around 1917 by the Islamic scholar Mohamed Turay, likely influenced both by the Vai syllabary and by Arabic. It was further developed over the next few years by his student and son-in-law Kisimi Kamara who added over 150 more syllabic characters, actively promoted the script, and is generally credited as its primary inventor. The repertoire is based on Tuchscherer 1996. Annotations in the names list provide occasional references to the syllabaries of Amara Mansaray, a prominent script practitioner, and David Dalby (Dalby 1967). The annotations note where Mansaray or Dalby vary from Tuchscherer.

Structure. The Mende Kikakui script has 185 consonant and vowel (CV) syllabic characters and 12 vowels. No script-specific punctuation is known.

Directionality. The Mende Kikakui script is written from right to left, unlike Vai. Conformant implementations of the script must use the Unicode Bidirectional Algorithm (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm").

Numbers. Although both European digits and Arabic digits have been used with Mende Kikakui, it also has its own unique non-decimal number system. This system uses the following characters:

- A set of digits one through nine
- A set of multiplier subscripts for powers of ten from 10 through 1,000,000, encoded as combining marks
- A special subscript for teens, also encoded as a combining mark

Number units in the range 11 through 19 are represented as a digit plus the teens mark. Numbers such as 20, 300, or 5,000 are represented as a digit plus the appropriate multiplier mark. Complete numbers are written as a right-to-left sequence of number units, largest unit first (displayed on the right), whose values are added to produce the numeric value, as shown in the examples in *Table 19-6*.

Table 19-6. Number Formation in Mende Kikakui

Value	Character Sequence	Display

10	1E8C7 MENDE KIKAKUI DIGIT ONE 1E8D1 MENDE KIKAKUI COMBINING NUMBER TENS	ጎ
14	1E8CA MENDE KIKAKUI DIGIT FOUR 1E8D0 MENDE KIKAKUI COMBINING NUMBER TEENS	ᒡ
27	1E8C8 MENDE KIKAKUI DIGIT TWO 1E8D1 MENDE KIKAKUI COMBINING NUMBER TENS 1E8CD MENDE KIKAKUI DIGIT SEVEN	ठ፞ᖟ
206	1E8C8 MENDE KIKAKUI DIGIT TWO 1E8D2 MENDE KIKAKUI COMBINING NUMBER HUNDREDS 1E8CC MENDE KIKAKUI DIGIT SIX	ठ̃ᒍ
417	1E8CA MENDE KIKAKUI DIGIT FOUR 1E8D2 MENDE KIKAKUI COMBINING NUMBER HUNDREDS 1E8CD MENDE KIKAKUI DIGIT SEVEN 1E8D0 MENDE KIKAKUI COMBINING NUMBER TEENS	ᒡ̃ᒍ
784	1E8CD MENDE KIKAKUI DIGIT SEVEN 1E8D2 MENDE KIKAKUI COMBINING NUMBER HUNDREDS 1E8CE MENDE KIKAKUI DIGIT EIGHT 1E8D1 MENDE KIKAKUI COMBINING NUMBER TENS 1E8CA MENDE KIKAKUI DIGIT FOUR	ᖟ̃ᖟᒍ

19.9 Adlam

19.9.1 Adlam: U+1E900–U+1E95F

Adlam is a script used to write Fulani and other African languages. The Fulani are a large, historically nomadic tribe of Africa numbering more than 45 million and spread across Senegambia (Senegal) to the banks of the Nile and the Red Sea. Depending on the language, they are called by different names, including Fulani, Fula, Peul, Pul, Fut, Fellata, Tekruri, Toucouleur, Peulh, Wasolonka, and Kourte.

The Fulani are today a widespread ethnic group in Africa, and the Fulani language is spoken by more than 40 million.

During the late 1980s, brothers Ibrahima and Abdoulaye Barry devised this alphabetic script to represent the Fulani language. After several years of development it was widely adopted among Fulani communities and is currently taught at schools in Guinea, Nigeria, Liberia and other nearby countries. The name *Adlam* is derived from the first four letters of the alphabet (A, D, L, M), standing for *Alkule Dandaydhe Leñol Mulugol* ("the alphabet that protects the peoples from vanishing").

Structure. Adlam is a casing script with right-to-left directionality. Its letters can be written separately or can be cursively joined in the same way that Arabic and N'Ko are. Joining is optional, not obligatory.

Diacritical Marks. A range of diacritical marks is used. The lengthener U+1E944 ADLAM ALIF LENGTHENER is used only on the letters U+1E900 ADLAM CAPITAL LETTER ALIF and U+1E922 ADLAM SMALL LETTER ALIF. The lengthener U+1E945 ADLAM VOWEL LENGTHENER is used with other vowels. The U+1E946 ADLAM GEMINATION MARK marks long consonants. These diacritical marks are typically high with capital letters, and high with small letters with ascenders, but low with other small letters.

The diacritical mark U+1E947 ADLAM HAMZA is used atop a consonant when a glottal stop occurs between it and the following vowel. The hamza has high and low variants. The mark U+1E948 ADLAM CONSONANT MODIFIER is used to indicate foreign sounds, primarily in Arabic transcription.

The U+1E94A ADLAM NUKTA is used to indicate both native and borrowed sounds. When vowels are lengthened, however, the nukta is drawn below the vowels to indicate the change. When drawn above a letter, the nukta is called *hoortobbhere* ("dot above") in Fulani; when drawn below, it is called *lestobbhere* ("dot below").

This varied rendering of the Adlam nukta is similar to the behavior of some accents in Latin typography, for which the rendering often depends on the availability of fonts, cultural preferences, or the geographical area. A Latin example is the prefer-

ence in Latvian and in Romanian for a comma below diacritic shape for some letters, while a cedilla shape is preferred for the same letters in Turkish and in Marshallese.

Line Breaking. Letters have the same line breaking behavior as N'Ko.

Numbers. Adlam uses ten digits with a right-to-left directionality like the digits in N'Ko.

Punctuation. Adlam uses European punctuation and the U+061F ARABIC QUES-TION MARK.

Cursive Joining. Cursive joining is used in some contexts. In a cursive context, all letters are dual-joining with a base form, a left-joining form, a dual-joining form, and a right-joining form. Diacritics do not break cursive connections.

Digits and punctuation do not participate in shaping. In a cursive context, U+0640 ARABIC TATWEEL can be used for elongation.

19.10 Medefaidrin

19.10.1 Medefaidrin: U+16E40–U+16E9F

The Medefaidrin script is used to write the liturgical language Medefaidrin by members of an indigenous Christian church, Oberi Okaime ("Church freely given"), which was active in the Nigerian province of Calabar in the 1930s near the Western bank of Cross River. The main spoken language for this group is Ibibio-Efik, which belongs to the Atlantic family of the Niger-Congo languages.

The Medefaidrin script shows the strong influence of English orthography with the use of capital and small letters, and a special sign for the pronoun "I", which has both an upper and lowercase form (U+16E44 MEDEFAIDRIN CAPITAL LETTER ATIU and U+16E64 MEDEFAIDRIN SMALL LETTER ATIU). The community tradition is that this spirit language was revealed to Bishop Ukpong, one of the founders of the community, in 1927 by divine inspiration. The secretary of the group, Jakeld Udofia, transcribed the language to writing. Presently, the religious community counts about 4,000 members. The Medefaidrin language is used for teaching Sunday school lessons and for saying prayers or meditation on the scriptures.

Structure. Medefaidrin is written left to right. There is a close relationship between the phonological analysis and the writing system: the letters are pronounced mostly as written.

Ordering. The order of Medefaidrin characters in the code charts reflects the traditional ordering of Medefaidrin found in instruction materials.

Punctuation, Digits, and Other Marks. Medefaidrin uses a vigesimal (base-20) number system that requires twenty digits. Script-specific punctuation marks are U+16E97 MEDEFAIDRIN COMMA, U+16E98 MEDEFAIDRIN FULL STOP, and U+16E9A MEDEFAIDRIN EXCLAMATION OH. Another unique mark is a symbol for the conjunction "or," represented by the Medefaidrin *aiva*, U+16E99 MEDEFAIDRIN SYMBOL AIVA.

19.11 Garay

19.11.1 Garay: U+10D40–U+10D8F

The Garay script was created by Assane Faye in Senegal to write Wolof, and first appeared in publications in January 1961. The official script in Senegal for Wolof is Latin, although the Wolofal orthography (an Arabic-based orthography written in the Ajami style) is also used. The Garay script has been taught informally for more than fifty years since its invention. However, although it is used by women's groups and adult literacy for Mandinka as well as for Wolof, the user community for Garay remains small. Faye designed Garay to be easy to learn for anyone familiar with the Arabic script, but gave it a simpler design. Faye has written manuscripts that include textbooks, folktales, and maps; there also exists a Quran with interlinear translations into Wolof using the Garay script.

Basic Features. Garay is an alphabet. Text runs from right to left in horizontal lines, and words are separated by spaces.

Garay has no joining behavior, but the consonant letters are bicameral. Capital letters are used at the beginning of sentences and for proper nouns. They are generally distinguished from lowercase letters by a swash. Unlike consonants, Garay vowel letters do not have uppercase and lowercase forms.

Tone does not need to be marked because languages in the Atlantic group of the Niger-Congo family, of which Wolof is one, are unusual in that they are not tonal.

A particular feature of Garay handwriting is a swash tail which often swings back under the word-final letter, and sometimes extends the whole length of the word, but it is not obligatory and has no semantic significance.

Consonants. Garay uses 19 consonant letters, each with both uppercase and lowercase forms, but digraphs and various diacritics over letters are used to support additional consonant sounds, including four pre-nasalized stops.

Consonant gemination is common and phonetically distinctive, and is written by following the consonant with U+10D6A ⦿ GARAY CONSONANT GEMINATION MARK. Another combining mark, U+10D6D ⦿ GARAY CONSONANT NASALIZATION MARK, can be used to nasalize sounds in foreign words.

To enable digitization of older Garay manuscripts, the Garay block contains four additional characters that represent obsolete glyphs for the letters K and N.

Vowels. Garay uses five characters, including one combining mark, to represent nine vowel sounds. The other four vowels are formed by combining the vowel characters into digraphs.

Long vowels are indicated by following the vowel character(s) with U+10D4E ◌ GARAY VOWEL LENGTH MARK.

The combining mark U+10D69 ◌ GARAY VOWEL SIGN E represents the vowel /ɛ/. It is also used to indicate prenasalized stops. When the vowel /ɛ/ follows prenasalized stops such as /ᵐb/ or other consonant letters with combining marks above, Garay instead uses the spacing mark U+10D4D ◌ GARAY VOWEL SIGN EE to represent that vowel. In the same context, the vowel /e/, normally represented as ◌◌ <U+10D69 ◌, U+10D4D ◌>, is written as ◌◌ <U+10D4D ◌, U+10D4D ◌>. This spelling approach is also used with the four consonants that *could* be prenasalized, /b/, /d/, /ɟ/, /g/, even when they are not.

Figure 19-4 illustrates these distinctions, with examples for the ordinary case with /k/ and then examples with /ᵐb/ and /b/.

Figure 19-4. Representing /ɛ/ and /e/ in Garay

Words that begin with a vowel sound use a "vowel carrier," U+10D70 ◌ GARAY SMALL LETTER A or its uppercase equivalent U+10D50 ◌ GARAY CAPITAL LETTER A, much like the *alef* in Arabic.

The Garay block also contains U+10D4F ◌ GARAY SUKUN to indicate vowel absence, but that character is now considered obsolete.

Punctuation. Garay uses a mixture of Western and Arabic punctuation.

The Garay block also includes U+10D6E ◌ GARAY HYPHEN, used at the end of a line when a word is split during line-breaking.

The symbol U+10D6F ◌ GARAY REDUPLICATION MARK, is used to repeat a word.

Numbers and Mathematical Symbols. A set of Garay digits is encoded in the range U+10D40..U+10D49. The block also contains Garay mathematical symbols for plus and minus signs.

Chapter 20

Americas

The Cherokee script is a syllabary developed between 1815 and 1821, to write the Cherokee language. The Cherokee script is still used by small communities in Oklahoma and North Carolina.

Canadian Aboriginal Syllabics were invented in the 1830s for Algonquian languages in Canada. The system has been extended many times, and is now actively used by other communities, including speakers of Inuktitut and Athapascan languages.

The Osage script is an alphabet used to write the Osage language spoken by a Native American tribe in the United States. The script was written with a variety of ad-hoc orthographies and transcriptions for two centuries until the Osage Nation recently developed its standard orthography in 2014.

Deseret is a phonemic alphabet devised in the 1850s to write English. It saw limited use for a few decades by members of The Church of Jesus Christ of Latter-day Saints.

20.1 Cherokee

20.1.1 Cherokee: U+13A0–U+13FF

Cherokee Supplement: U+AB70–U+ABBF

The Cherokee script is used to write the Cherokee language. Cherokee is a member of the Iroquoian language family. It is related to Cayuga, Seneca, Onondaga, Wyandot-Huron, Tuscarora, Oneida, and Mohawk. The relationship is not close because roughly 3,000 years ago the Cherokees migrated southeastward from the Great Lakes region of North America to what is now North Carolina, Tennessee, and Georgia. Cherokee is the native tongue of approximately 20,000 people, although most speakers today use it as a second language. The Cherokee word for both the language and the people is ᏣᎳᎩ *Tsalagi*.

The Cherokee syllabary, as invented by Sequoyah between 1815 and 1821, contained 6 vowels and 17 consonants. Sequoyah avoided copying from other alphabets, but his original letters were modified to make them easier to print. Samuel Worcester worked in conjunction with Sequoyah, Chief Charles Hicks, and Charles Thompson (first cousin of Sequoyah) in the design of the Cherokee type which was finalized in 1827. Using fonts available to him, Worcester assigned a number of Latin letters to the Cherokee syllables. At this time the Cherokee letter "MV" was dropped, and the Cherokee syllabary reached the size of 85 letters. Worcester's press printed 13,980,000 pages of Native American-language text, most of it in Cherokee.

Structure. Cherokee is a left-to-right script. It has no Cherokee-specific combining characters.

Casing. Most existing Cherokee text is caseless. Traditionally, the forms of the syllable letters were designed as caps height—and in fact, a number of the Cherokee syllables are visually indistinguishable from Latin uppercase letters. As a result, most Cherokee text has the visual appearance of all caps. The characters used for representing such unicameral Cherokee text are the basic syllables in the Cherokee block: U+13A0 CHEROKEE LETTER A, and so forth.

In some old printed material, such as the Cherokee New Testament, case conventions adapted from the Latin script were used. Sentence-initial letters and initial letters for personal and place names, for example, were typeset using a larger size font. Furthermore, systematic distinction in casing has become more prevalent in modern typeset materials, as well.

Starting with Version 8.0, the Unicode Standard includes a set of lowercase Cherokee syllables to accommodate the need to represent casing distinctions in Cherokee text. The Cherokee script is now encoded as a fully bicameral script, with case mapping.

The lowercase syllable letters are mostly encoded in the Cherokee Supplement block. A few are encoded at the end of the Cherokee block, after the basic Cherokee syllable letters, which are now treated as the uppercase of the case pairs.

The usual way for a script originally encoded in the Unicode Standard as a unicameral script to later gain casing is by adding a new set of uppercase letters for it. The Cherokee script is an important exception because the previously encoded Cherokee unicameral set is treated as the uppercase as of Version 8.0, and the new set of letters are the lowercase. The reason for this exception has to do with Cherokee typography and the status of existing fonts. Because all existing fonts already treated Cherokee syllable letters as cap height, attempting to extend them by changing the existing letters to less than cap height and adding new uppercase letters to the fonts would have destabilized the layout of all existing Cherokee text. On the other hand, innovating in the fonts by adding new lowercase forms with a smaller size and less than cap height allows a graceful introduction of casing without invalidating the layout of existing text.

This exceptional introduction of a lowercase set to change a unicameral encoding into a bicameral encoding has important implications that implementers of the Cherokee script need to keep in mind. First, in order to preserve case folding stability, Cherokee case folds to the previously encoded uppercase letters, rather than to the newly encoded lowercase letters. This exceptional case folding behavior impacts identifiers, and so can trip up implementations if they are not prepared for it. Second, representation of cased Cherokee text requires using the new lowercase letters for most of the body text, instead of just changing a few initial letters to uppercase. That means that representation of traditional text such as the Cherokee New Testament requires substantial re-encoding of the text. Third, the fact that *uppercase* Cherokee still represents the default and is most widely supported in fonts means that input systems which are extended to support the new lowercase letters face unusual design choices.

Tones. Each Cherokee syllable can be spoken on one of four pitch or tone levels, or can slide from one pitch to one or two others within the same syllable. However, only in certain words does the tone of a syllable change the meaning. Tones are unmarked.

Input. Several keyboarding conventions exist for inputting Cherokee. Some involve dead-key input based on Latin transliterations; some are based on sound-mnemonics related to Latin letters on keyboards; and some are ergonomic systems based on frequency of the syllables in the Cherokee language

Numbers. Although Sequoyah invented a Cherokee number system, it was not adopted and is not encoded in the Unicode Standard. The Cherokee Nation uses European numbers. Cherokee speakers pay careful attention to the use of ordinal and cardinal numbers. When speaking of a numbered series, they will use ordinals. For

example, when numbering chapters in a book, Cherokee headings would use First Chapter, Second Chapter, and so on, instead of Chapter One, Chapter Two, and so on.

Punctuation. Cherokee uses standard Latin punctuation.

Standards. There are no other encoding standards for Cherokee. Cherokee spelling is not standardized: each person spells as the word sounds to him or her.

20.2 Canadian Aboriginal Syllabics

20.2.1 Unified Canadian Aboriginal Syllabics: U+1400–U+167F

The characters in this block are a unification of various local syllabaries of Canada into a single repertoire based on character appearance. The syllabics were invented in the late 1830s by James Evans for Algonquian languages. As other communities and linguistic groups adopted the script, the main structural principles described in this section were adopted. The primary user community for this script consists of several aboriginal groups throughout Canada, including Algonquian, Inuktitut, and Athapascan language families. The script is also used by governmental agencies and in business, education, and media.

Organization. The repertoire is organized primarily on structural principles found in the CASEC [1994] report, and is essentially a glyphic encoding. The canonical structure of each character series consists of a consonant shape with five variants. Typically the shape points down when the consonant is combined with the vowel /e/, up when combined with the vowel /i/, right when combined with the vowel /o/, and left when combined with the vowel /a/. It is reduced and superscripted when in syllable-final position, not followed by a vowel. For example:

∨	∧	>	<	ᐤ
PE	PI	PO	PA	P

Some variations in vowels also occur. For example, in Inuktitut usage, the syllable U+1450 ᑐ CANADIAN SYLLABICS TO is transcribed into Latin letters as "TU" rather than "TO", but the structure of the syllabary is generally the same regardless of language.

Arrangement. The arrangement of signs follows the Algonquian ordering (down-pointing, up-pointing, right-pointing, left-pointing), as in the previous example.

Sorted within each series are the variant forms for that series. Algonquian variants appear first, then Inuktitut variants, then Athapascan variants. This arrangement is convenient and consistent with the historical diffusion of Syllabics writing; it does not imply any hierarchy.

Some glyphs do not show the same down/up/right/left directions in the typical fashion—for example, beginning with U+146B ᑫ CANADIAN SYLLABICS KE. These glyphs are variations of the rule because of the shape of the basic glyph; they do not affect the convention.

Vowel length and labialization modify the character series through the addition of various marks (for example, U+143E ᐾ CANADIAN SYLLABICS PWII). Such modified

characters are considered unique syllables. They are not decomposed into base characters and one or more diacritics. Some language families have different conventions for placement of the modifying mark. For the sake of consistency and simplicity, and to support multiple North American languages in the same document, each of these variants is assigned a unique code point.

Carrier Syllabics. The Carrier syllabics orthography has been unified with the Unified Canadian Aboriginal Syllabics blocks, but many characters are unique. Carrier syllabics differ from more widely-used systems like Cree and Inuktitut. Carrier syllables have a square proportion and are uniform in width and height, and the syllable finals are vertically centered rather than superscripted, as shown in *Figure 20-1*. Font designers need to pay particular attention to these distinctions in pan-Syllabics fonts that try to cover the entire Unified Canadian Aboriginal Syllabics character repertoire. Using the Cree or Inuktitut proportions for Carrier syllabics would look very wrong to Carrier users.

Figure 20-1. Position of Carrier Syllable Finals
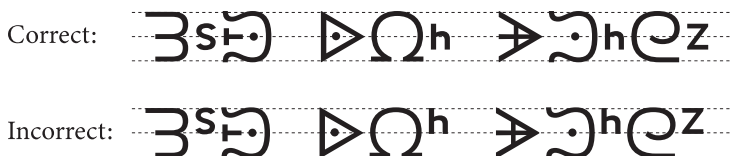
Extensions. A few additional syllables in the range U+166F..U+167F at the end of this block have been added for Inuktitut, Woods Cree, and Blackfoot. Because these extensions were encoded well after the main repertoire in the block, their arrangement in the code charts is outside the framework for the rest of the characters in the block.

Punctuation and Symbols. Languages written using the Canadian Aboriginal Syllabics make use of the common punctuation marks of Western typography. However, a few punctuation marks are specific in form and are separately encoded as script-specific marks for syllabics. These include: U+166E CANADIAN SYLLABICS FULL STOP and U+1400 CANADIAN SYLLABICS HYPHEN.

There is also a special symbol, U+166D CANADIAN SYLLABICS CHI SIGN, used in religious texts as a symbol to denote Christ.

20.2.2 Unified Canadian Aboriginal Syllabics Extended: U+18B0– U+18FF

This block contains many additional syllables attested in various local traditions of syllabics usage in Canada. These additional characters include extensions for several Algonquian communities (Cree, Moose Cree, and Ojibway), and for several Dene communities (Beaver Dene, Hare Dene, Chipewyan, and Carrier).

20.2.3 Unified Canadian Aboriginal Syllabics Extended-A: U+11AB0–U+11ABF

This block contains twelve syllables at U+11AB0..U+11ABB needed for rendering unique sounds in the Nattilik language (Nattilingmiutut) used in Western Nunavut, a member of the Western Canadian Inuktitut language family. These syllables improve the support for Nattilik provided in other blocks. This block also includes four historic syllables at U+11ABC..U+11ABF used in early documents for the Cree and Ojibway languages.

20.3 Osage

20.3.1 Osage: U+104B0–U+104FF

The Osage script is used to write the Osage language. This language is spoken by a Native American tribe of the Great Plains that originated in the Ohio River valley area of the present-day United States. By the 17th century, the Osage people had migrated to their current locations in Missouri, Kansas, Arkansas, Oklahoma, and Texas. The term "Osage" roughly translates to "mid-waters."

For two centuries, the Osage language was written with a variety of ad-hoc Latin orthographies and transcription systems. In 2004, the Osage Nation initiated a program to develop a standard orthography to write the language. By 2006, a practical orthography had been designed based on modifications or fusions of the shapes of Latin letters. Use of the Osage orthography led to further improvements, culminating in the adoption of the current set of letters in 2014.

Structure. Osage is a left-to-right alphabetic script. It has no Osage-specific combining characters, but makes use of common diacritical marks.

Casing. Casing is used in the standard Osage orthography.

Vowels. Diacritical marks are used in Osage to distinguish length, nasalization, and accents. The particular diacritical marks used to make these distinctions are shown in *Table 20-1*.

Table 20-1. Combining Marks used in Osage

Nasal vowels	U+0358 ◌̘ COMBINING DOT ABOVE RIGHT
Long vowels	U+0304 ◌̄ COMBINING MACRON
Pitch accents	U+0301 ◌́ COMBINING ACUTE ACCENT
Pitch accent with vowel length	U+030B ◌̋ COMBINING DOUBLE ACUTE ACCENT

Numbers and Punctuation. Osage uses European numbers and standard Latin punctuation.

20.4 Deseret

20.4.1 Deseret: U+10400–U+1044F

Deseret is a phonemic alphabet devised to write the English language. It was originally developed in the 1850s by the regents of the University of Deseret, now the University of Utah. It was promoted by The Church of Jesus Christ of Latter-day Saints, also known as the "Mormon" or LDS Church, under Church President Brigham Young (1801–1877). The name *Deseret* is taken from a word in the Book of Mormon defined to mean "honeybee" and reflects the LDS use of the beehive as a symbol of cooperative industry. Most literature about the script treats the term *Deseret Alphabet* as a proper noun and capitalizes it as such.

Among the designers of the Deseret Alphabet was George D. Watt, who had been trained in shorthand and served as Brigham Young's secretary. It is possible that, under Watt's influence, Sir Isaac Pitman's 1847 English Phonotypic Alphabet was used as the model for the Deseret Alphabet.

The Deseret Alphabet was a work in progress through most of the 1850s, with the set of letters and their shapes changing from time to time. The final version was used for the printed material of the late 1860s, but earlier versions are found in handwritten manuscripts.

The Church commissioned two typefaces and published four books using the Deseret Alphabet. The Church-owned *Deseret News* also published passages of scripture using the alphabet on occasion. In addition, some historical records, diaries, and other materials were handwritten using this script, and it had limited use on coins and signs. There is also one tombstone in Cedar City, Utah, written in the Deseret Alphabet. However, the script failed to gain wide acceptance and was not actively promoted after 1869. Today, the Deseret Alphabet remains of interest primarily to historians and hobbyists.

Letter Names and Shapes. Pedagogical materials produced by the LDS Church gave names to all of the non-vowel letters and indicated the vowel sounds with English examples. In the Unicode Standard, the spelling of the non-vowel letter names has been modified to clarify their pronunciations, and the vowels have been given names that emphasize the parallel structure of the two vowel runs.

The glyphs used in the Unicode Standard are derived from the second typeface commissioned by the LDS Church and represent the shapes most commonly encountered. Alternate glyphs are found in the first typeface and in some instructional material.

Structure. The final version of the script consists of 38 letters, LONG I through ENG. Two additional letters, OI and EW, found only in handwritten materials, are encoded after the first 38. The alphabet is bicameral; capital and small letters differ only in size and not in shape. The order of the letters is phonetic: letters for similar classes of sound are grouped together. In particular, most consonants come in unvoiced/voiced pairs. Forty-letter versions of the alphabet inserted OI after AY and EW after OW.

Sorting. The order of the letters in the Unicode Standard is the one used in all but one of the nineteenth-century descriptions of the alphabet. The exception is one in which the letters WU and YEE are inverted. The order YEE-WU follows the order of the "coalescents" in Pitman's work; the order WU-YEE appears in a greater number of Deseret materials, however, and has been followed here.

Alphabetized material followed the standard order of the Deseret Alphabet in the code charts, except that the short and long vowel pairs are grouped together, in the order long vowel first, and then short vowel.

Typographic Conventions. The Deseret Alphabet is written from left to right. Punctuation, capitalization, and digits are the same as in English. All words are written phonemically with the exception of short words that have pronunciations equivalent to letter names, as shown in *Figure 20-2*.

Figure 20-2. Short Words Equivalent to Deseret Letter Names

𐐰	AY is written for *eye* or *I*
𐐏	YEE is written for *ye*
𐐁	BEE is written for *be* or *bee*
𐐅	GAY is written for *gay*
𐐒	THEE is written for *the* or *thee*

Phonetics. An approximate IPA transcription of the sounds represented by the Deseret Alphabet is shown in *Table 20-2*.

Table 20-2. IPA Transcription of Deseret

Deseret	Name	IPA	Deseret	Name	IPA
𐐀𐐀	LONG I	i	𐐁𐐁	BEE	b
𐐂𐐂	LONG E	e	𐐃𐐃	TEE	t
𐐄𐐄	LONG A	ɑ	𐐆𐐆	DEE	d
𐐈𐐈	LONG AH	ɒ	𐐉𐐉	CHEE	tʃ
𐐊𐐊	LONG O	o	𐐋𐐋	JEE	dʒ
𐐌𐐌	LONG OO	u	𐐍𐐍	KAY	k
𐐎𐐎	SHORT I	ɪ	𐐏𐐏	GAY	g
𐐐𐐐	SHORT E	ɛ	𐐑𐐑	EF	f
𐐒𐐒	SHORT A	æ	𐐓𐐓	VEE	v
𐐔𐐔	SHORT AH	ɔ	𐐕𐐕	ETH	θ
𐐖𐐖	SHORT O	ʌ	𐐗𐐗	THEE	ð
𐐘𐐘	SHORT OO	ʊ	𐐙𐐙	ES	s
𐐚𐐚	AY	ɑɪ	𐐛𐐛	ZEE	z
𐐜𐐜	OI	ɔɪ	𐐝𐐝	ESH	ʃ
𐐞𐐞	OW	ɑʊ	𐐟𐐟	ZHEE	ʒ
𐐠𐐠	EW	ju	𐐡𐐡	ER	r
𐐢𐐢	WU	w	𐐣𐐣	EL	l
𐐤𐐤	YEE	j	𐐥𐐥	EM	m
𐐦𐐦	H	h	𐐧𐐧	EN	n
𐐨𐐨	PEE	p	𐐩𐐩	ENG	ŋ

Chapter 21

Notational Systems

Braille consists of a related set of notational systems, using raised dots embossed on paper or other mediums to provide a tactile writing system for the blind. The patterns of dots are associated with the letters or syllables of other writing systems, but the particular rules of association vary from language to language. The Unicode Standard encodes a complete set of symbols for the shapes of Braille patterns; however the association of letters to the patterns is left to other standards. Text should normally be represented using the regular Unicode characters of the script. Only when the intent is to convey a particular binding of text to a Braille pattern sequence should it be represented using the symbols for the Braille patterns.

Musical notation—particularly Western musical notation—is different from ordinary text in the way it is laid out, especially the representation of pitch and duration in Western musical notation. However, ordinary text commonly refers to the basic graphical elements that are used in musical notation, so such symbols are encoded in the Unicode Standard. Additional sets of symbols for Ancient Greek, Byzantine, and Znamenny notation are encoded to support historical systems of musical notation.

Duployan is an uncased, alphabetic stenographic writing system invented by Emile Duployé, and published in 1860. It was one of the two most commonly used French shorthands. The Duployan shorthands are used as a secondary shorthand for writing French, English, German, Spanish, and Romanian. An adaptation and augmentation of Duployan was used as an alternate primary script for several First Nations' languages in interior British Columbia, Canada.

Sutton SignWriting is a notational system developed in 1974 by Valerie Sutton and used for the transcription of many sign languages. It is a featural writing system, in which visually iconic basic symbols are arranged in two-dimensional layout to form snapshots of the individual signs of a sign language, which are roughly equivalent to words. The Unicode Standard encodes the basic symbols as atomic characters or combining character sequences.

21.1 Braille

21.1.1 Braille Patterns: U+2800–U+28FF

Braille is a writing system used by blind people worldwide. It uses a system of six or eight raised dots, arranged in two vertical rows of three or four dots, respectively. Eight-dot systems build on six-dot systems by adding two extra dots above or below the core matrix. Six-dot Braille allows 64 possible combinations, and eight-dot Braille allows 256 possible patterns of dot combinations. There is no fixed correspondence between a dot pattern and a character or symbol of any given script. Dot pattern assignments are dependent on context and user community. A single pattern can represent an abbreviation or a frequently occurring short word. For a number of contexts and user communities, the series of ISO technical reports starting with ISO/ TR 11548-1 provide standardized correspondence tables as well as invocation sequences to indicate a context switch.

The Unicode Standard encodes a single complete set of 256 eight-dot patterns. This set includes the 64 dot patterns needed for six-dot Braille.

The character names for Braille patterns are based on the assignments of the dots of the Braille pattern to digits 1 to 8 as follows:

$$
\begin{array}{ccc}
1 & \bullet\,\bullet & 4 \\
2 & \bullet\,\bullet & 5 \\
3 & \bullet\,\bullet & 6 \\
7 & \bullet\,\bullet & 8 \\
\end{array}
$$

The designation of dots 1 to 6 corresponds to that of six-dot Braille. The additional dots 7 and 8 are added beneath. The character name for a Braille pattern consists of BRAILLE PATTERN DOTS-12345678, where only those digits corresponding to dots in the pattern are included. The name for the empty pattern is BRAILLE PATTERN BLANK.

The 256 Braille patterns are arranged in the same sequence as in ISO/TR 11548-1, which is based on an octal number generated from the pattern arrangement. Octal numbers are associated with each dot of a Braille pattern in the following way:

$$
\begin{array}{ccc}
1 & \bullet\,\bullet & 10 \\
2 & \bullet\,\bullet & 20 \\
4 & \bullet\,\bullet & 40 \\
100 & \bullet\,\bullet & 200 \\
\end{array}
$$

The octal number is obtained by adding the values corresponding to the dots present in the pattern. Octal numbers smaller than 100 are expanded to three digits by inserting leading zeroes. For example, the dots of BRAILLE PATTERN DOTS-1247 are

assigned to the octal values of 1_8, 2_8, 10_8, and 100_8. The octal number representing the sum of these values is 113_8.

The assignment of meanings to Braille patterns is outside the scope of this standard.

Example. According to ISO/TR 11548-2, the character LATIN CAPITAL LETTER F can be represented in eight-dot Braille by the combination of the dots 1, 2, 4, and 7 (BRAILLE PATTERN DOTS-1247). A full circle corresponds to a tangible (set) dot, and empty circles serve as position indicators for dots not set within the dot matrix:

$$1 \; ●● \; 4$$
$$2 \; ●○ \; 5$$
$$3 \; ○○ \; 6$$
$$7 \; ●○ \; 8$$

Usage Model. The eight-dot Braille patterns in the Unicode Standard are intended to be used with either style of eight-dot Braille system, whether the additional two dots are considered to be in the top row or in the bottom row. These two systems are never intermixed in the same context, so their distinction is a matter of convention. The intent of encoding the 256 Braille patterns in the Unicode Standard is to allow input and output devices to be implemented that can interchange Braille data without having to go through a context-dependent conversion from semantic values to patterns, or vice versa. In this manner, final-form documents can be exchanged and faithfully rendered. At the same time, processing of textual data that require semantic support is intended to take place using the regular character assignments in the Unicode Standard.

Imaging. When output on a Braille device, dots shown as black are intended to be rendered as tangible. Dots shown in the standard as open circles are blank (not rendered as tangible). The Unicode Standard does not specify any physical dimension of Braille characters.

In the absence of a higher-level protocol, Braille patterns are output from left to right. When used to render final form (tangible) documents, Braille patterns are normally not intermixed with any other Unicode characters except control codes.

Script. Unlike other sets of symbols, the Braille Patterns are given their own, unique value of the Script property in the Unicode Standard. This follows both from the behavior of Braille in forming a consistent writing system on its own terms, as well as from the independent bibliographic status of books and other documents printed in Braille. For more information on the Script property, see Unicode Standard Annex #24, "Unicode Script Property."

21.2 Western Musical Symbols

21.2.1 Musical Symbols: U+1D100–U+1D1FF

The musical symbols encoded in the Musical Symbols block are intended to cover basic Western musical notation and its antecedents: mensural notation and plainsong (or Gregorian) notation, as well as closely related systems, such as Kievan notation. The most comprehensive coded language in regular use for representing sound is the common musical notation (CMN) of the Western world. Western musical notation is a system of symbols that is relatively, but not completely, self-consistent and relatively stable but still, like music itself, evolving. This open-ended system has survived over time partly because of its flexibility and extensibility. In the Unicode Standard, musical symbols have been drawn primarily from CMN. Commonly recognized additions to the CMN repertoire, such as quarter-tone accidentals, cluster noteheads, and shape-note noteheads, have also been included.

Graphical score elements are not included in the Musical Symbols block. These pictographs are usually created for a specific repertoire or sometimes even a single piece. Characters that have some specialized meaning in music but that are found in other character blocks are not included. They include numbers for time signatures and figured basses, letters for section labels and Roman numeral harmonic analysis, and so on.

Musical symbols are used worldwide in a more or less standard manner by a very large group of users. The symbols frequently occur in running text and may be treated as simple spacing characters with no special properties, with a few exceptions. Musical symbols are used in contexts such as theoretical works, pedagogical texts, terminological dictionaries, bibliographic databases, thematic catalogs, and databases of musical data. The musical symbol characters are also intended to be used within higher-level protocols, such as music description languages and file formats for the representation of musical data and musical scores.

Because of the complexities of layout and of pitch representation in general, the encoding of musical pitch is intentionally outside the scope of the Unicode Standard. The Musical Symbols block provides a common set of elements for interchange and processing. Encoding of pitch, and layout of the resulting musical structure, involves specifications not only for the vertical relationship between multiple notes simultaneously, but also in multiple staves, between instrumental parts, and so forth. These musical features are expected to be handled entirely in higher-level protocols making use of the graphical elements provided. Lack of pitch encoding is not a shortcoming, but rather is a necessary feature of the encoding.

Glyphs. The glyphs for musical symbols shown in the code charts, are representative of typical cases; however, note in particular that the stem direction is not specified by

the Unicode Standard and can be determined only in context. For a font that is intended to provide musical symbols in running text, either stem direction is acceptable. In some contexts—particularly for applications in early music—note heads, stems, flags, and other associated symbols may need to be rendered in different colors —for example, red.

Symbols in Other Blocks. U+266D MUSIC FLAT SIGN, U+266E MUSIC NATURAL SIGN, and U+266F MUSIC SHARP SIGN—three characters that occur frequently in musical notation—are encoded in the Miscellaneous Symbols block (U+2600..U+267F). However, four characters also encoded in that block are to be interpreted merely as dingbats or miscellaneous symbols, not as representing actual musical notes:

U+2669 QUARTER NOTE

U+266A EIGHTH NOTE

U+266B BEAMED EIGHTH NOTES

U+266C BEAMED SIXTEENTH NOTES

Processing. Most musical symbols can be thought of as simple spacing characters when used inline within texts and examples, even though they behave in a more complex manner in full musical layout. Some characters are meant only to be combined with others to produce combined character sequences, representing musical notes and their particular articulations. Musical symbols can be input, processed, and displayed in a manner similar to mathematical symbols. When embedded in text, most of the symbols are simple spacing characters with no special properties. A few characters have format control functions, as described later in this section.

Input Methods. Musical symbols can be entered via standard alphanumeric keyboard, via piano keyboard or other device, or by a graphical method. Keyboard input of the musical symbols may make use of techniques similar to those used for Chinese, Japanese, and Korean. In addition, input methods utilizing pointing devices or piano keyboards could be developed similar to those in existing musical layout systems. For example, within a graphical user interface, the user could choose symbols from a palette-style menu.

Directionality. When combined with right-to-left texts—in Hebrew or Arabic, for example—the musical notation is usually written from left to right in the normal manner. The words are divided into syllables and placed under or above the notes in the same fashion as for Latin and other left-to-right scripts. The individual words or syllables corresponding to each note, however, are written in the dominant direction of the script.

The opposite approach is also known: in some traditions, the musical notation is actually written from right to left. In that case, some of the symbols, such as clef signs, are mirrored; other symbols, such as notes, flags, and accidentals, are *not* mirrored. All responsibility for such details of bidirectional layout lies with higher-level protocols and is not reflected in any character properties. *Figure 21-1* exemplifies this principle with two musical passages. The first example shows Turkish lyrics in Arabic script with ordinary left-to-right musical notation; the second shows right-to-left musical notation. Note the partial mirroring.

Figure 21-1. Examples of Specialized Music Layout

Format Characters. Extensive ligature-like beams are used frequently in musical notation between groups of notes having short values. The practice is widespread and very predictable, so it is therefore amenable to algorithmic handling. The format characters U+1D173 MUSICAL SYMBOL BEGIN BEAM and U+1D174 MUSICAL SYMBOL END BEAM can be used to indicate the extents of beam groupings. In some exceptional cases, beams are left unclosed on one end. This status can be indicated with a U+1D159 MUSICAL SYMBOL NULL NOTEHEAD character if no stem is to appear at the end of the beam.

Similarly, format characters have been provided for other connecting structures. The characters U+1D175 MUSICAL SYMBOL BEGIN TIE, U+1D176 MUSICAL SYMBOL END TIE, U+1D177 MUSICAL SYMBOL BEGIN SLUR, U+1D178 MUSICAL SYMBOL END SLUR, U+1D179 MUSICAL SYMBOL BEGIN PHRASE, and U+1D17A MUSICAL SYMBOL END PHRASE indicate the extent of these features. Like beaming, these features are easily handled in an algorithmic fashion.

These pairs of characters modify the layout and grouping of notes and phrases in full musical notation. When musical examples are written or rendered in plain text without special software, the start/end format characters may be rendered as brackets or left uninterpreted. To the extent possible, more sophisticated software that renders musical examples inline with natural-language text might interpret them in their actual format control capacity, rendering slurs, beams, and so forth, as appropriate.

Precomposed Note Characters. For maximum flexibility, the character set includes both precomposed note values and primitives from which complete notes may be constructed. The precomposed versions are provided mainly for convenience. However, if any normalization form is applied, including NFC, the characters will be decomposed. For further information, see *Section 3.11, Normalization Forms*. The canonical equivalents for these characters are given in the Unicode Character Database and are illustrated in *Figure 21-2*.

Figure 21-2. Precomposed Note Characters
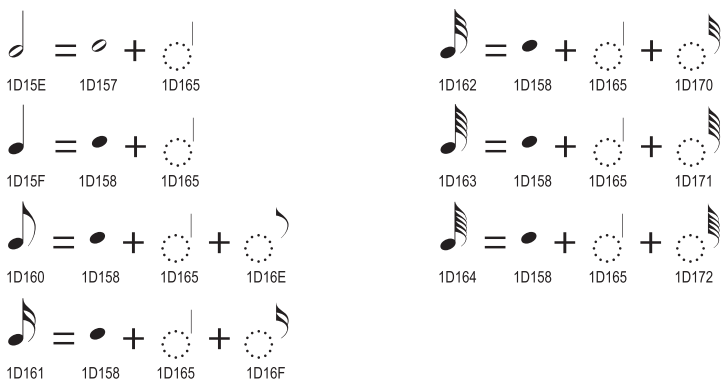
Alternative Noteheads. More complex notes built up from alternative noteheads, stems, flags, and articulation symbols are necessary for complete implementations and complex scores. Examples of their use include American shape-note and modern percussion notations, as shown in the first line of *Figure 21-3*.

Figure 21-3. Alternative Noteheads
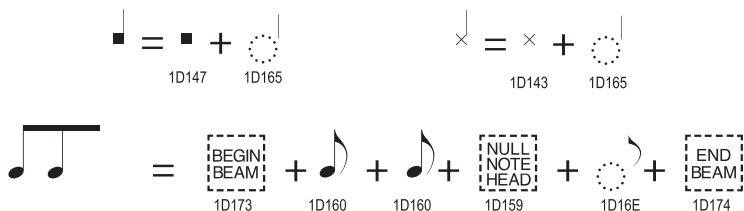
U+1D159 MUSICAL SYMBOL NULL NOTEHEAD is a special notehead that has no distinct visual appearance of its own. It can be used as an anchor for a combining flag in complicated musical scoring. For example, in a beamed sequence of notes, the beam might be extended beyond visible notes, as shown in the second line of *Figure 21-3*. Even though the *null notehead* has no visual appearance of its own, it is not a default ignorable code point; some indication of its presence, as for instance a dotted box glyph, should be shown if displayed outside of a context that supports full musical rendering.

Augmentation Dots and Articulation Symbols. Augmentation dots and articulation symbols may be appended to either the precomposed or built-up notes. In addition, augmentation dots and articulation symbols may be repeated as necessary to build a complete note symbol. Examples of the use of augmentation dots and articulation symbols are shown in *Figure 21-4*.

Figure 21-4. Augmentation Dots and Articulation Symbols
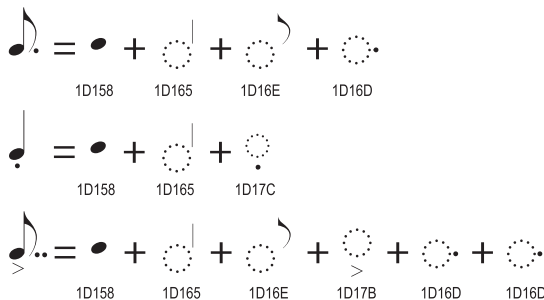
Ornamentation. *Table 21-1* lists common eighteenth-century ornaments and the sequences of characters from which they can be generated.

Table 21-1. Examples of Ornamentation

⁓	1D19C STROKE-2 + 1D19D STROKE-3
⁓	1D19C STROKE-2 + 1D1A0 STROKE-6 + 1D19D STROKE-3
⁓	1D1A0 STROKE-6 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3
⁓	1D19C STROKE-2 + 1D19C STROKE-2 + 1D1A0 STROKE-6 + 1D19D STROKE-3
⁓	1D19C STROKE-2 + 1D19C STROKE-2 + 1D1A3 STROKE-9
⁓	1D1A1 STROKE-7 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3
⁓	1D1A2 STROKE-8 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3
⁓	1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3 + 1D19F STROKE-5
⁓	1D1A1 STROKE-7 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D1A0 STROKE-6 + 1D19D STROKE-3
⁓	1D1A1 STROKE-7 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3 + 1D19F STROKE-5
⁓	1D1A2 STROKE-8 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D1A0 STROKE-6 + 1D19D STROKE-3
⁓	1D19B STROKE-1 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3
⁓	1D19B STROKE-1 + 1D19C STROKE-2 + 1D19C STROKE-2 + 1D19D STROKE-3 + 1D19E STROKE-4
⁓	1D19C STROKE-2 + 1D19D STROKE-3 + 1D19E STROKE-4

Gregorian. The *punctum*, or Gregorian *brevis*, a square shape, is unified with U+1D147 MUSICAL SYMBOL SQUARE NOTEHEAD BLACK. The Gregorian *semibrevis*, a diamond or lozenge shape, is unified with U+1D1BA MUSICAL SYMBOL SEMIBREVIS

BLACK. Thus Gregorian notation, medieval notation, and modern notation either require separate fonts in practice or need font features to make subtle differentiations between shapes where required.

Kievan. Kievan musical notation is a form of linear musical notation found in religious chant books of the Russian Orthodox Church, among others. It is also referred to as East Slavic musical notation. The notation originated in the 1500s, and the first books using Kievan notation were published in 1772. The notation is still used today.

Unlike Western plainchant, Kievan is written on a five-line staff (encoded at U+1D11A) with uniquely shaped notes, and several distinct symbols, including its own C clef and flat signs. U+1D1DF MUSICAL SYMBOL KIEVAN END OF PIECE is analogous to the Western U+1D102 MUSICAL SYMBOL FINAL BARLINE.

Beaming is used in Kievan notation occasionally, and the existing musical format characters encoded between U+1D173 and U+1D17A may be used in implementations of beaming in higher-level protocols.

Persian. Persian traditional music uses intervals that are approximately equivalent to a quarter-tone, but which are not equal-tempered. The 20th-century composer Ali-Naqi Vaziri introduced two symbols, called *sori* and *koron*, to represent these intervals. They are encoded as U+1D1E9 ♯ MUSICAL SYMBOL SORI and U+1D1EA ♭ MUSICAL SYMBOL KORON. The *sori* is analogous to U+1D132 ♯ MUSICAL SYMBOL QUARTER TONE SHARP, while the *koron* is analogous to U+1D133 ♭ MUSICAL SYMBOL QUARTER TONE FLAT.

21.3 Byzantine Musical Symbols

21.3.1 Byzantine Musical Symbols: U+1D000–U+1D0FF

Byzantine musical notation first appeared in the seventh or eighth century CE, developing more fully by the tenth century. These musical symbols are chiefly used to write the religious music and hymns of the Christian Orthodox Church, although folk music manuscripts are also known. In 1881, the Orthodox Patriarchy Musical Committee redefined some of the signs and established the New Analytical Byzantine Musical Notation System, which is in use today. About 95% of the more than 7,000 musical manuscripts using this system are in Greek. Other manuscripts are in Russian, Bulgarian, Romanian, and Arabic.

Processing. Computer representation of Byzantine musical symbols is quite recent, although typographic publication of religious music books began in 1820. Two kinds of applications have been developed: applications to enable musicians to write the books they use, and applications that compare or convert this musical notation system to the standard Western system. (See *Section 21.2, Western Musical Symbols*.)

Byzantine musical symbols are divided into 15 classes according to function. Characters interact with one another in the horizontal and vertical dimension. There are three horizontal "stripes" in which various classes generally appear and rules as to how other characters interact within them. These rules, which are still being specified, are the responsibilities of higher-level protocols.

21.4 Znamenny Musical Notation

21.4.1 Znamenny Musical Notation: U+1CF00–U+1CFCF

Znamenny musical notation is used to write Znamenny chant, a form of liturgical singing that developed in Russia in the 11th century CE. Znamenny chant was the predominant form of liturgical music used in Russia and Ukraine until the late 17th century. After that time, Russian Old Ritualists, as well as some monasteries and parishes within the mainline Russian Orthodox Church, continued to use Znamenny musical notation.

While Znamenny chant has limited modern use within the Russian Orthodox Church, musicologists and liturgists began academic research into Znamenny chant in the 19th century, and this research continues today. Derived from an early form of Byzantine musical notation, Znamenny notation developed over five centuries, and came to form a unique notation system. Notably, Znamenny notation does not use a lined staff. In Znamenny notation, neumes are a note or a group of notes to be sung to a single syllable.

Classification. Modern Znamemmy notation has three varieties: types A, B, and C. The earliest is *Type C* notation, which occurs in musical manuscripts from the 15th century onward and lacks any markings indicating pitch. *Type B* notation arose in the first half of the 17th century, when special marks indicating pitch and dynamics were introduced. Historically, these marks were made in red ink, so they were called Cinnabar or Shaidur marks. This block in the Unicode Standard primarily encodes the system of Cinnabar marks documented in the 1670 treatise *Izveshchenie o soglasneyshikh pometakh.*

Priznaki. In the late 17th century Znamenny notation needed to be typeset on the newly developed printing press. Because the available type technology did not allow simultaneous printing of neumes in black and red ink, a monochrome system of alternate pitch marks was devised using small dashes, called *priznaki*, to indicate pitch. This system came to be used alongside the Cinnabar marks in a unified writing system. Notation bearing both the priznaki and Cinnabar marks is called *Type A* notation.

21.5 Ancient Greek Musical Notation

21.5.1 Ancient Greek Musical Notation: U+1D200–U+1D24F

Ancient Greeks developed their own distinct system of musical notation, which is found in a large number of ancient texts ranging from a fragment of Euripides' *Orestes* to Christian hymns. It is also used in the modern publication of these texts as well as in modern studies of ancient music.

The system covers about three octaves, and symbols can be grouped by threes: one symbol corresponds to a "natural" note on a diatonic scale, and the two others to successive sharpenings of that first note. There is no distinction between enharmonic and chromatic scales. The system uses two series of symbols: one for vocal melody and one for instrumental melody.

The symbols are based on Greek letters, comparable to the modern usage of the Latin letters A through G to refer to notes of the Western musical scale. However, rather than using a sharp and flat notation to indicate semitones, or casing and other diacritics to indicate distinct octaves, the Ancient Greek system extended the basic Greek alphabet by rotating and flipping letterforms in various ways and by adding a few more symbols not directly based on letters.

Unification. In the Unicode Standard, the vocal and instrumental systems are unified with each other and with the basic Greek alphabet, based on shape. *Table 21-2* gives the correspondence between modern notes, the numbering used by modern scholars, and the Unicode characters or sequences of characters to use to represent them.

Table 21-2. Representation of Ancient Greek Vocal and Instrumental

Modern Note	Modern Number	Vocal Notation	Instrumental Notation
g″	70	2127, 0374	1D23C, 0374
	69	0391, 0374	1D23B, 0374
	68	0392, 0374	1D23A, 0374
f″	67	0393, 0374	039D, 0374
	66	0394, 0374	1D239, 0374
	65	0395, 0374	1D208, 0374
e″	64	0396, 0374	1D238, 0374
	63	0397, 0374	1D237, 0374
	62	0398, 0374	1D20D, 0374
d″	61	0399, 0374	1D236, 0374
	60	039A, 0374	1D235, 0374
	59	039B, 0374	1D234, 0374

c″	58	039C, 0374	1D233, 0374
	57	039D, 0374	1D232, 0374
	56	039E, 0374	1D20E, 0374
b′	55	039F, 0374	039A, 0374
	54	1D21C	1D241
	53	1D21B	1D240
a′	52	1D21A	1D23F
	51	1D219	1D23E
	50	1D218	1D23D
g′	49	2127	1D23C
	48	0391	1D23B
	47	0392	1D23A
f′	46	0393	039D
	45	0394	1D239
	44	0395	1D208
e′	43	0396	1D238
	42	0397	1D237
	41	0398	1D20D
d′	40	0399	1D236
	39	039A	1D235
	38	039B	1D234
c′	37	039C	1D233
	36	039D	1D232
	35	039E	1D20E
b	34	039F	039A
	33	03A0	03FD
	32	03A1	1D231
a	31	03F9	03F9
	30	03A4	1D230
	29	03A5	1D22F
g	28	03A6	1D213
	27	03A7	1D22E
	26	03A8	1D22D
f	25	03A9	1D22C
	24	1D217	1D22B
	23	1D216	1D22A
e	22	1D215	0393
	21	1D214	1D205
	20	1D213	1D21C

d	19	1D212	1D229
	18	1D211	1D228
	17	1D210	1D227
c	16	1D20F	0395
	15	1D20E	1D211
	14	1D20D	1D226
B	13	1D20C	1D225
	12	1D20B	1D224
	11	1D20A	1D223
A	10	1D209	0397
	9	1D208	1D206
	8	1D207	1D222
G	7	1D206	1D221
	6	1D205	03A4
	5	1D204	1D220
F	4	1D203	1D21F
	3	1D202	1D202
	2	1D201	1D21E
E	1	1D200	1D21D

Naming Conventions. The character names are based on the standard names widely used by modern scholars. There is no standardized ancient system for naming these characters. Apparent gaps in the numbering sequence are due to the unification with standard letters and between vocal and instrumental notations.

If a symbol is used in both the vocal notation system and the instrumental notation system, its Unicode character name is based on the vocal notation system catalog number. Thus U+1D20D GREEK VOCAL NOTATION SYMBOL-14 has a glyph based on an inverted capital lambda. In the vocal notation system, it represents the first sharp of B; in the instrumental notation system, it represents the first sharp of d'. Because it is used in both systems, its name is based on its sequence in the vocal notation system, rather than its sequence in the instrumental notation system. The character names list in the Unicode Character Database is fully annotated with the functions of the symbols for each system.

Font. Scholars usually typeset musical characters in sans-serif fonts to distinguish them from standard letters, which are usually represented with a serifed font. However, this is not required. The code charts use a font without serifs for reasons of clarity.

Combining Marks. The combining marks encoded in the range U+1D242..U+1D244 are placed over the vocal or instrumental notation symbols. They are used to indicate metrical qualities.

21.6 Duployan

21.6.1 Duployan: U+1BC00–U+1BC9F

The Duployan shorthands are used to write French, English, German, Spanish, and Romanian. The original Duployan shorthand was invented by Emile Duployé, and published in 1860 as a stenographic shorthand for French. It was one of the two most commonly used French shorthands. There are three main English adaptations from the late 19th and early 20th centuries based on Duployan: Pernin, Sloan, and Perrault. None were as popular as the Gregg and Pitman shorthands.

An adaptation and augmentation of Duployan by Father Jean Marie Raphael LeJeune was used as an alternate primary script for several First Nations' languages in interior British Columbia, including Chinook Jargon, Okanagan, Lilooet, Shushwap, and North Thompson. Its original use and greatest surviving attestation is from the *Kamloops Wawa*, a Chinook Jargon newsletter of the Catholic diocese of Kamloops, British Columbia, published 1891–1923. Chinook Jargon was a trade language widely spoken from southeast Alaska to northern California, from the Pacific to the Rockies, and sporadically outside this area. The Chinook script uses the basic Duployan inventory, with the addition of several derived letterforms and compound letters.

Structure. Duployan is an uncased, alphabetic stenographic writing system. The model letterforms are generally based on circles and lines. It is a left-to-right script.

The basic inventory of consonant and vowel signs has been augmented over the years to provide more efficient shorthands and has been adapted to the phonologies of languages other than the original French. The Romanian Pernin, Perrault, and Sloan stenographic orthographies add a few letters or letterforms, ideographs, and several combined letters.

The core repertoire of Duployan contains several classes of letters, differentiated primarily by visual form and stroke direction, and nominally by phonetic value. Letter classes include the line consonants (P, T, F, K, and L-type), arc consonants (M, N, J, and S-type), circle vowels (A and O vowels), nasal vowels, and orienting vowels (U/EU, I/E). In addition, the Chinook writing contains spacing letters, compound consonants, and a logograph.

The extended Duployan shorthand includes four other letter classes—the complex letters (multisyllabic symbols with consonant forms), and high, low, and connecting terminals for common word endings. The repertoire also includes U+1BC9D DUPLOYAN THICK LETTER SELECTOR, which modifies a preceding Duployan character by causing it to be rendered bold.

For further details and discussion of implementation of rendering for Duployan, see Unicode Technical Note #37, "Duployan Shorthand Rendering Model."

Representative Glyphs. The representative glyphs used in the Unicode code charts for Duployan characters often include additional information about direction of strokes and/or relative position for connecting terminals. In particular, for letters that are differentiated by stroke direction, small arrows are placed next to the glyphs for those letters in the code charts, to indicate that the stroke direction is upwards or downwards, for example. These small arrows are intended to help identify and distinguish such letter pairs, and would not be included as part of glyphs in fonts for rendering connected Duployan text. In a similar manner, for some attached affixes, the representative glyphs are shown together with dotted lines that indicate contrasts in the relative position of their attachment, but which are not displayed in rendered text.

21.6.2 Shorthand Format Controls: U+1BCA0–U+1BCAF

Many systems of shorthand use overlapping letters to indicate abbreviations and initialisms. (Initialisms are abbreviations that are pronounced one letter at a time, such as IBM or HTML.) Such non-default text flow may be controlled with the shorthand format controls. U+1BCA0 SHORTHAND FORMAT LETTER OVERLAP indicates a single letter overlap, with the text continuing to flow as if that overlapping character did not exist. U+1BCA1 SHORTHAND FORMAT CONTINUING OVERLAP indicates a continuing overlap where the text flow proceeds from the overlapping character. In Duployan, the overlapping behavior is limited to consonants, circle vowels, and orienting vowels overlapping consonants.

There are two other "step" format controls used with word endings and contractions in specific contexts. U+1BCA2 SHORTHAND FORMAT DOWN STEP indicates downstep, which means that a following character should be rendered below the previous character, with any subsequent joined characters proceeding relative to the lowered glyph. U+1BCA3 SHORTHAND FORMAT UP STEP indicates upstep, which causes the following word or stenographic full stop to be raised.

21.7 Sutton SignWriting

21.7.1 Sutton SignWriting: U+1D800–U+1DAAF

Sutton SignWriting is a notational system developed in 1974 by Valerie Sutton and used for the transcription of many sign languages. It is designed to represent physical formations of sign language signs precisely, and is used in a number of publications. More information about the notational system and catalogs of signs can be found on the Sutton SignWriting websites http://www.signwriting.org/ and http://www.signbank.org/.

Structure. Sutton SignWriting is a featural writing system, in which visually iconic basic symbols are arranged in two-dimensional layout to form snapshots of the individual signs of a sign language, which are roughly equivalent to words. The Unicode Standard encodes the basic symbols as atomic characters or combining character sequences. The spatial arrangement of the symbols is an essential part of the writing system, but constitutes a higher-level protocol beyond the scope of the Unicode Standard.

Repertoire. The repertoire of Sutton SignWriting is comprised of characters for handshapes, which are the configurations that the hands take in signing, as well as characters for contact, movement, head and face, body, and location. The repertoire also includes five punctuation marks and twenty characters that indicate fill and rotation.

The head and face characters are used in combining character sequences to represent facial expressions. The character sequences are formed with U+1D9FF SIGNWRITING HEAD as base, followed by nonspacing marks from the ranges U+1DA00..U+1DA36 and U+1DA3B..U+1DA6C. These nonspacing marks represent expressions or movements of the eyes, cheeks, mouth, and so on, and include such characters as U+1DA17 SIGNWRITING EYE BLINK SINGLE and U+1DA3E SIGNWRITING MOUTH SMILE.

Modifiers. The fill and rotation characters are nonspacing combining marks that modify a base character to create various realizations of the base character. For example, the handshape U+1D800 SIGNWRITING HAND-FIST INDEX can be modified by a fill character, a rotation character, or both to represent different positions of that handshape and to distinguish between the left and the right hand.

There are five fill modifiers, U+1DA9B SIGNWRITING FILL MODIFIER-2 through U+1DA9F SIGNWRITING FILL MODIFIER-6, and fifteen rotation modifiers, U+1DAA1 SIGNWRITING ROTATION MODIFIER-2 through U+1DAAF SIGNWRITING ROTATION MODIFIER-16. There are no explicit modifiers encoded for *fill-1* or *rotation-1*, as those values are considered inherent in the base character. When both a fill and a rotation

modifier are used in a combining character sequence, the fill modifier precedes the rotation modifier in the sequence.

The effect of a fill modifier depends on the character sequence it appears in. For example, when applied to a handshape character such as U+1D800 SIGNWRITING HAND-FIST INDEX, a fill modifier selects one of six possible fills representing as many palm orientations. When applied to a tempo symbol such as U+1D9F7 SIGNWRITING DYNAMIC FAST, a fill modifier alters the shape of the base character. When used in a character sequence such as <U+1D9FF SIGNWRITING HEAD, U+1DA16 SIGN-WRITING EYES CLOSED, *fill*>, the fill modifier selects between one eye and both eyes closed.

The rotation modifiers turn a base character by 45 degree increments. In combination with handshape characters, the rotation modifiers also distinguish between the right and left hand characters. U+1DAA4 SIGNWRITING ROTATION MODIFIER-5 turns a base character by 180 degrees. For a handshape that distinguishes between right and left hand shapes, U+1DAAC SIGNWRITING ROTATION MODIFIER-13 turns the left hand shape 180 degrees.

Punctuation. Sutton SignWriting uses five script-specific punctuation marks. These include U+1DA8B SIGNWRITING PARENTHESIS, which represents an opening paren-thesis. A closing parenthesis is represented with the sequence <U+1DA8B SIGN-WRITING PARENTHESIS, U+1DAA4 SIGNWRITING ROTATION MODIFIER-5>.

Chapter 22

Symbols

The universe of symbols is rich and open-ended.

Pictorial or graphic items for which there is no demonstrated need or strong desire to exchange in plain text are not encoded in the standard.

Combining marks may be used with symbols, particularly the set encoded at U+20D0..U+20FF (see *Section 7.9, Combining Marks*).

Letterlike and currency symbols, as well as numerals, superscripts, and subscripts, are typically subject to the same font and style changes as the surrounding text. Where square and enclosed symbols occur in East Asian contexts, they generally follow the prevailing type styles.

Other symbols have an appearance that is independent of type style, or a more limited or altogether different range of type style variation than the regular text surrounding them. For example, mathematical alphanumeric symbols are typically used for mathematical variables; those letterlike symbols that are part of this set carry semantic information in their type style. This fact restricts—but does not completely eliminate—possible style variations. However, symbols such as mathematical operators can be used with any script or independent of any script.

Special invisible operator characters can be used to explicitly encode some mathematical operations, such as multiplication, which are normally implied by juxtaposition. This aids in automatic interpretation of mathematical notation.

In a bidirectional context (see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm"), most symbol characters have no inherent directionality but resolve their directionality for display according to the Unicode Bidirectional Algorithm. For some symbols, such as brackets and mathematical operators whose image is not bilaterally symmetric, the mirror image is used when the character is part of the right-to-left text stream (see *Section 4.7, Bidi Mirrored*).

Dingbats and optical character recognition characters are different from all other characters in the standard, in that they are encoded based primarily on their precise appearance.

Many symbols encoded in the Unicode Standard are intended to support legacy implementations and obsolescent practices, such as terminal emulation or other character mode user interfaces. Examples include box drawing components and control pictures.

A number of symbols are also encoded for emoji ("picture character," or pictograph). Added initially for compatibility with the emoji sets encoded by several Japanese cell phone carriers as extensions of the JIS X 0208 character set, these pictographs continue to grow in usage and coverage. These symbols are interchanged as plain text, and are encoded in the Unicode Standard to support interoperability and widespread usage on mobile devices.

Other symbols—many of which are also pictographic—are encoded for compatibility with Webdings and Wingdings sets, or various e-mail systems, and to address other interchange requirements.

Many of the symbols encoded in Unicode can be used as operators or given some other syntactical function in a formal language syntax. For more information, see Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax."

22.1 Currency Symbols

Currency symbols are intended to encode the customary symbolic signs used to indicate certain currencies in general text. These signs vary in shape and are often used for more than one currency. Not all currencies are represented by a special currency symbol; some use multiple-letter strings instead, such as "Sfr" for Swiss franc. Moreover, the abbreviations for currencies can vary by language. The Unicode Common Locale Data Repository (CLDR) provides further information; see https://cldr.unicode.org. Therefore, implementations that are concerned with the *exact* identity of a currency should not depend on an encoded currency sign character. Instead, they should follow standards such as the ISO 4217 three-letter currency codes, which are *specific* to currencies—for example, USD for U.S. dollar, CAD for Canadian dollar.

Unification. The Unicode Standard does not duplicate encodings where more than one currency is expressed with the same symbol. Many currency symbols are overstruck letters. There are therefore many minor variants, such as the U+0024 DOLLAR SIGN $, with one or two vertical bars, or other graphical variation, as shown in *Figure 22-1*.

Figure 22-1. Alternative Glyphs for Dollar Sign

$ $

Claims that glyph variants of a certain currency symbol are used consistently to indicate a particular currency could not be substantiated upon further research. Therefore, the Unicode Standard considers these variants to be typographical and provides a single encoding for them. See ISO/IEC 10367, Annex B (informative), for an example of multiple renderings for U+00A3 POUND SIGN.

Fonts. Currency symbols are commonly designed to display at the same width as a digit (most often a European digit, U+0030..U+0039) to assist in alignment of monetary values in tabular displays. Like letters, they tend to follow the stylistic design features of particular fonts because they are used often and need to harmonize with body text. In particular, even though there may be more or less normative designs for the currency sign per se, as for the euro sign, type designers freely adapt such designs to make them fit the logic of the rest of their fonts. This partly explains why currency signs show more glyph variation than other types of symbols.

22.1.1 Currency Symbols: U+20A0–U+20CF

This block contains currency symbols that are not encoded in other blocks. Contemporary and historic currency symbols encoded in other blocks are listed in *Table 22-1*.

The table omits currency symbols known only from usage in ancient coinage, such as U+1017A GREEK TALENT SIGN and U+10196 ROMAN DENARIUS SIGN.

Table 22-1. Currency Symbols Encoded in Other Blocks

Currency	Unicode Code Point	
Dollar, milreis, escudo, peso	U+0024	DOLLAR SIGN
Cent	U+00A2	CENT SIGN
Pound and lira	U+00A3	POUND SIGN
General currency	U+00A4	CURRENCY SIGN
Yen or yuan	U+00A5	YEN SIGN
Dutch florin	U+0192	LATIN SMALL LETTER F WITH HOOK
Dram	U+058F	ARMENIAN DRAM SIGN
Afghani	U+060B	AFGHANI SIGN
Rupee	U+09F2	BENGALI RUPEE MARK
Rupee	U+09F3	BENGALI RUPEE SIGN
Ana (historic)	U+09F9	BENGALI CURRENCY DENOMINATOR SIXTEEN
Ganda (historic)	U+09FB	BENGALI GANDA MARK
Rupee	U+0AF1	GUJARATI RUPEE SIGN
Rupee	U+0BF9	TAMIL RUPEE SIGN
Baht	U+0E3F	THAI CURRENCY SYMBOL BAHT
Riel	U+17DB	KHMER CURRENCY SYMBOL RIEL
German mark (historic)	U+2133	SCRIPT CAPITAL M
Yuan, yen, won, HKD	U+5143	CJK UNIFIED IDEOGRAPH-5143
Yen	U+5186	CJK UNIFIED IDEOGRAPH-5186
Yuan	U+5706	CJK UNIFIED IDEOGRAPH-5706
Yuan, yen, won, HKD, NTD	U+5713	CJK UNIFIED IDEOGRAPH-5713
Rupee	U+A838	NORTH INDIC RUPEE MARK
Rial	U+FDFC	RIAL SIGN

Lira Sign. A separate currency sign U+20A4 LIRA SIGN is encoded for compatibility with the HP Roman-8 character set, which is still widely implemented in printers. In general, U+00A3 POUND SIGN may be used for both the various currencies known as pound (or punt) and the currencies known as lira. Examples include the British pound sterling, the historic Irish punt, and the former lira currency of Italy. Until 2012, the lira sign was also used for the Turkish lira, but for current Turkish usage, see U+20BA TURKISH LIRA SIGN. As in the case of the dollar sign, the glyphic distinction between single- and double-bar versions of the sign is not indicative of a systematic difference in the currency.

Dollar and Peso. The dollar sign (U+0024) is used for many currencies in Latin America and elsewhere. In particular, this use includes current and discontinued

Latin American peso currencies, such as the Mexican, Chilean, Colombian and Dominican pesos. However, the Philippine peso uses a different symbol found at U+20B1.

Yen and Yuan. Like the dollar sign and the pound sign, U+00A5 YEN SIGN has been used as the currency sign for more than one currency. The double-crossbar glyph is the official form for both the yen currency of Japan (JPY) and for the yuan (renminbi) currency of China (CNY). This is the case, despite the fact that some glyph standards historically specified a single-crossbar form, notably the OCR-A standard ISO 1073-1:1976, which influenced the representative glyph in various character set standards from China. In the Unicode Standard, U+00A5 YEN SIGN is intended to be the character for the currency sign for both the yen and the yuan, independent of the details of glyphic presentation.

As listed in _Table 22-1_, there are also a number of CJK ideographs to represent the words *yen* (or *en*) and *yuan*, as well as the Korean word *won*, and these also tend to overlap in use as currency symbols.

Euro Sign. The single currency for member countries of the European Economic and Monetary Union is the euro (EUR). The euro character is encoded in the Unicode Standard as U+20AC EURO SIGN.

Indian Rupee Sign. U+20B9 ₹ INDIAN RUPEE SIGN is the character encoded to represent the Indian rupee currency symbol introduced by the Government of India in 2010 as the official currency symbol for the Indian rupee (INR). It is distinguished from U+20A8 RUPEE SIGN, which is an older symbol not formally tied to any particular currency. There are also a number of script-specific rupee symbols encoded for historic usage by various scripts of India. See _Table 22-1_ for a listing.

Rupee is also the common name for a number of currencies for other countries of South Asia and of Indonesia, as well as several historic currencies. It is often abbreviated using Latin letters, or may be spelled out or abbreviated in the Arabic script, depending on local conventions.

Turkish Lira Sign. The Turkish lira sign, encoded as U+20BA ₺ TURKISH LIRA SIGN, is a symbol representing the lira currency of Turkey. Prior to the introduction of the new symbol in 2012, the currency was typically abbreviated with the letters "TL". The new symbol was selected by the Central Bank of Turkey from entries in a public contest and is quickly gaining common use, but the old abbreviation is also still in use.

Ruble Sign. The ruble sign, encoded as U+20BD ₽ RUBLE SIGN, was adopted as the official symbol for the currency of Russian Federation in 2013. Ruble is also used as the name of various currencies in Eastern Europe. In English, both spellings "ruble" and "rouble" are used.

Lari Sign. The lari sign, encoded as U+20BE ₾ LARI SIGN, was adopted as the official symbol for the currency of Georgia in 2014. The name *lari* is an old Georgian word denoting a hoard or property. The image for the lari sign is based on the letter U+10DA ლ GEORGIAN LETTER LAS. The lari currency was established on October 2, 1995.

Bitcoin Sign. U+20BF ₿ BITCOIN SIGN represents the *bitcoin*, a cryptocurrency and payment system invented by programmers. A cryptocurrency such as the bitcoin works as a medium of exchange that uses cryptography to secure transactions and to control the creation of additional units of currency. It is categorized as a decentralized virtual or digital currency.

Som Sign. U+20C0 ⃀ SOM SIGN was adopted as the official currency symbol of the Kyrgyz Republic on February 8, 2017. The som currency was introduced with bank notes on May 10, 1993 to replace the Soviet ruble. Coins were added later in 2008.

Other Currency Symbols. Additional forms of currency symbols are found in the Small Form Variants (U+FE50..U+FE6F) and the Halfwidth and Fullwidth Forms (U+FF00..U+FFEF) blocks. Those symbols have the General_Category property value Currency_Symbol (gc = Sc).

Ancient Greek and Roman monetary symbols, for such coins and values as the Greek *obol* or the Roman *denarius* and *as*, are encoded in the Ancient Greek Numbers (U+10140..U+1018F) and Ancient Symbols (U+10190..U+101CF) blocks. Those symbols denote values of weights and currencies, but are not used as regular currency symbols. As such, their General_Category property value is Other_Symbol (gc = So).

22.2 Letterlike Symbols

22.2.1 Letterlike Symbols: U+2100–U+214F

Letterlike symbols are symbols derived in some way from ordinary letters of an alphabetic script. This block includes symbols based on Latin, Greek, and Hebrew letters. Stylistic variations of single letters are used for semantics in mathematical notation. See "Mathematical Alphanumeric Symbols" in this section for the use of letterlike symbols in mathematical formulas. Some letterforms have given rise to specialized symbols, such as U+211E PRESCRIPTION TAKE.

Numero Sign. U+2116 NUMERO SIGN is provided both for Cyrillic use, where it looks like №, and for compatibility with Asian standards, where it looks like №. *Figure 22-2* illustrates a number of alternative glyphs for this sign. Instead of using a special symbol, French practice is to use an "N" or an "n", according to context, followed by a superscript small letter "o" (N^o or n^o; plural N^{os} or n^{os}). Legacy data encoded in ISO/IEC 8859-1 (Latin-1) or other 8-bit character sets may also have represented the *numero sign* by a sequence of "N" followed by the *degree sign* (U+00B0 DEGREE SIGN). Implementations interworking with legacy data should be aware of such alternative representations for the *numero sign* when converting data.

Figure 22-2. Alternative Glyphs for Numero Sign

№ № №. №

Unit Symbols. Several letterlike symbols are used to indicate units. In most cases, however, such as for SI units (Système International), the use of regular letters or other symbols is preferred. U+2113 SCRIPT SMALL L is commonly used as a non-SI symbol for the *liter*. Official SI usage prefers the regular *lowercase letter l*.

Three letterlike symbols have been given canonical equivalence to regular letters: U+2126 OHM SIGN, U+212A KELVIN SIGN, and U+212B ANGSTROM SIGN. In all three instances, the regular letter should be used. If text is normalized according to Unicode Standard Annex #15, "Unicode Normalization Forms," these three characters will be replaced by their regular equivalents.

In normal use, it is better to represent degrees Celsius "°C" with a sequence of U+00B0 DEGREE SIGN + U+0043 LATIN CAPITAL LETTER C, rather than U+2103 DEGREE CELSIUS. For searching, treat these two sequences as identical. Similarly, the sequence U+00B0 DEGREE SIGN + U+0046 LATIN CAPITAL LETTER F is preferred over U+2109 DEGREE FAHRENHEIT, and those two sequences should be treated as identical for searching.

Compatibility. Some symbols are composites of several letters. Many of these composite symbols are encoded for compatibility with Asian and other legacy encodings. (See also "CJK Compatibility Ideographs" in *Section 18.1, Han*.) The use of these composite symbols is discouraged where their presence is not required by compatibility. For example, in normal use, the symbols U+2121 TEL TELEPHONE SIGN and U+213B FAX FACSIMILE SIGN are simply spelled out.

In the context of East Asian typography, many letterlike symbols, and in particular composites, form part of a collection of compatibility symbols, the larger part of which is located in the CJK Compatibility block (see *Section 22.10, Enclosed and Square*). When used in this way, these symbols are rendered as "wide" characters occupying a full cell. They remain upright in vertical layout, contrary to the rotated rendering of their regular letter equivalents. See Unicode Standard Annex #11, "East Asian Width," for more information.

Where the letterlike symbols have alphabetic equivalents, they collate in alphabetic sequence; otherwise, they should be treated as symbols. The letterlike symbols may have different directional properties than normal letters. For example, the four transfinite cardinal symbols (U+2135..U+2138) are used in ordinary mathematical text and do not share the strong right-to-left directionality of the Hebrew letters from which they are derived.

Styles. The letterlike symbols include some of the few instances in which the Unicode Standard encodes stylistic variants of letters as distinct characters. For example, there are instances of blackletter (*Fraktur*), double-struck, italic, and script styles for certain Latin letters used as mathematical symbols. The choice of these stylistic variants for encoding reflects their common use as distinct symbols. They form part of the larger set of mathematical alphanumeric symbols. For the complete set and more information on its use, see "Mathematical Alphanumeric Symbols" in this section. These symbols should not be used in ordinary, nonscientific texts.

Despite its name, U+2118 SCRIPT CAPITAL P is neither script nor capital—it is uniquely the Weierstrass elliptic function symbol derived from a calligraphic *lowercase* p. U+2113 SCRIPT SMALL L is derived from a special *italic* form of the *lowercase letter l* and, when it occurs in mathematical notation, is known as the symbol *ell*. Use U+1D4C1 MATHEMATICAL SCRIPT SMALL L as the *lowercase script l* for mathematical notation.

Standards. The Unicode Standard encodes letterlike symbols from many different national standards and corporate collections.

22.2.2 Mathematical Alphanumeric Symbols: U+1D400–U+1D7FF

The Mathematical Alphanumeric Symbols block contains a large extension of letterlike symbols used in mathematical notation, typically for variables. The characters in

this block are intended for use only in mathematical or technical notation, and not in nontechnical text. When used with markup languages—for example, with Mathematical Markup Language (MathML)—the characters are expected to be used directly, instead of indirectly via entity references or by composing them from base letters and style markup.

Words Used as Variables. In some specialties, whole words are used as variables, not just single letters. For these cases, style markup is preferred because in ordinary mathematical notation the juxtaposition of variables generally implies multiplication, not word formation as in ordinary text. Markup not only provides the necessary scoping in these cases, but also allows the use of a more extended alphabet.

22.2.3 Mathematical Alphabets

Basic Set of Alphanumeric Characters. Mathematical notation uses a basic set of mathematical alphanumeric characters, which consists of the following:

- The set of basic Latin digits (0–9) (U+0030..U+0039)
- The set of basic uppercase and lowercase Latin letters (a–z, A–Z)
- The uppercase Greek letters A–Ω (U+0391..U+03A9), plus the nabla ∇ (U+2207) and the variant of theta Θ given by U+03F4
- The lowercase Greek letters α–ω (U+03B1..U+03C9), plus the partial differential sign ∂ (U+2202), and the six glyph variants ε, ϑ, ϰ, ϕ, ϱ, and ϖ, given by U+03F5, U+03D1, U+03F0, U+03D5, U+03F1, and U+03D6, respectively

Only unaccented forms of the letters are used for mathematical notation, because general accents such as the acute accent would interfere with common mathematical diacritics. Examples of common mathematical diacritics that can interfere with general accents are the circumflex, macron, or the single or double dot above, the latter two of which are used in physics to denote derivatives with respect to the time variable. Mathematical symbols with diacritics are always represented by combining character sequences.

For some characters in the basic set of Greek characters, two variants of the same character are included. This is because they can appear in the same mathematical document with different meanings, even though they would have the same meaning in Greek text. (See "Variant Letterforms" in *Section 7.2, Greek*.)

Additional Characters. In addition to this basic set, mathematical notation uses the uppercase and lowercase digamma, in regular (U+03DC and U+03DD) and bold (U+1D7CA and U+1D7CB), and the four Hebrew-derived characters (U+2135..U+2138). Occasional uses of other alphabetic and numeric characters are known. Examples include U+0428 CYRILLIC CAPITAL LETTER SHA, U+306E HIRA-GANA LETTER NO, and Eastern Arabic-Indic digits (U+06F0..U+06F9). However,

these characters are used only in their basic forms, rather than in multiple mathematical styles.

Dotless Characters. In the Unicode Standard, the characters "i" and "j", including their variations in the mathematical alphabets, have the Soft_Dotted property. Any conformant renderer will remove the dot when the character is followed by a nonspacing combining mark above. Therefore, using an individual mathematical italic *i* or *j* with math accents would result in the intended display. However, in mathematical equations an entire sub-expression can be placed underneath a math accent —for example, when a "wide hat" is placed on top of *i+j*, as shown in <u>Figure 22-3</u>.

Figure 22-3. Wide Mathematical Accents

$$\widehat{i+j} = \hat{i} + \hat{j}$$

In such a situation, a renderer can no longer rely simply on the presence of an adjacent combining character to substitute for the un-dotted glyph, and whether the dots should be removed in such a situation is no longer predictable. Authors differ in whether they expect the dotted or dotless forms in that case.

In some documents *mathematical italic dotless i or j* is used explicitly without any combining marks, or even in contrast to the dotted versions. Therefore, the Unicode Standard provides the explicitly dotless characters U+1D6A4 MATHEMATICAL ITALIC SMALL DOTLESS I and U+1D6A5 MATHEMATICAL ITALIC SMALL DOTLESS J. These two characters map to the ISOAMSO entities *imath* and *jmath* or the T$_E$X macros \imath and \jmath. These entities are, by default, always italic. The appearance of these two characters in the code charts is similar to the shapes of the entities documented in the ISO 9573-13 entity sets and used by T$_E$X. The mathematical dotless characters do not have case mappings.

Semantic Distinctions. Mathematical notation requires a number of Latin and Greek alphabets that initially appear to be mere font variations of one another. The letter H can appear as plain or upright (H), bold (**H**), italic (*H*), as well as script, Fraktur, and other styles. However, in any given document, these characters have distinct, and usually unrelated, mathematical semantics. For example, a normal H represents a different variable from a bold **H**, and so on. If these attributes are dropped in plain text, the distinctions are lost and the meaning of the text is altered. Without the distinctions, the well-known Hamiltonian formula turns into the *integral* equation in the variable H as shown in <u>Figure 22-4</u>.

Figure 22-4. Style Variants and Semantic Distinctions in Mathematics

$$\text{Hamiltonian formula:} \quad \mathcal{H} = \int d\tau(\epsilon E^2 + \mu H^2)$$

$$\text{Integral equation:} \quad H = \int d\tau(\varepsilon E^2 + \mu H^2)$$

Mathematicians will object that a properly formatted integral equation requires all the letters in this example (except for the "d") to be in italics. However, because the distinction between $\mathcal{H}$ and H has been lost, they would recognize it as a fallback representation of an integral equation, and not as a fallback representation of the Hamiltonian. By encoding a separate set of alphabets, it is possible to preserve such distinctions in plain text.

Mathematical Alphabets. The sets of distinctly styled mathematical alphanumeric symbols are listed in <u>Table 22-2</u>.

Table 22-2. Mathematical Alphanumeric Symbols

Math Style	Characters from Basic Set	Location
plain (upright, serifed)	Latin, Greek, and digits	BMP
bold	Latin, Greek, and digits	Plane 1
italic	Latin and Greek	Plane 1
bold italic	Latin and Greek	Plane 1
script (calligraphic)	Latin	Plane 1/BMP
bold script (calligraphic)	Latin	Plane 1
Fraktur	Latin	Plane 1
bold Fraktur	Latin	Plane 1
double-struck	Latin and digits	Plane 1
sans-serif	Latin and digits	Plane 1
sans-serif bold	Latin, Greek, and digits	Plane 1
sans-serif italic	Latin	Plane 1
sans-serif bold italic	Latin and Greek	Plane 1
monospace	Latin and digits	Plane 1

The styles in <u>Table 22-2</u> represent those encountered in mathematical use. The plain letters have been unified with the existing characters in the Basic Latin and Greek blocks. There are 24 double-struck, italic, Fraktur, and script characters that already exist in the Letterlike Symbols block (U+2100..U+214F). These are explicitly unified with the characters in this block, and corresponding holes have been left in the mathematical alphabets.

The alphabets encoded in the Mathematical Alphanumeric Symbols block on Plane 1 represent the distinctions between different mathematically styled semantic alphabets, but the exact glyphs shown in the code charts are not intended to be prescriptive for actual mathematical font design. In particular, the script and double-struck styles show considerable variation across mathematical fonts.

Characters from the Mathematical Alphanumeric symbols block should *not* be used to represent styling of nonmathematical text.

Script Style and Calligraphic Variants. The mathematical script style, also referred to as the *calligraphic* style, has two widely recognized, specific variant styles: *chancery*, as exemplified by the glyph $\mathcal{L}$, and *roundhand*, as exemplified by the glyph $\mathscr{L}$. In most mathematical documents, the chancery calligraphic style and the roundhand calligraphic style are considered interchangeable. Accordingly, when the mathematical alphanumeric symbols were added to the Unicode Standard in Version 3.1, those two styles were unified. However, documentation subsequently emerged demonstrating that the regular (non-bold) uppercase Latin script characters occasionally show semantic contrasts between chancery style and roundhand style. To accommodate this usage, variation sequences have been defined, starting with Version 14.0, to distinguish chancery and roundhand variants. These variation sequences work as follows:

> An uppercase mathematical script style letter followed by U+FE00 displays in chancery style.

> An uppercase mathematical script style letter followed by U+FE01 displays in roundhand style.

Otherwise, an uppercase mathematical script style letter will display with the default for the font in use. The exact list of defined variation sequences can be found in the StandardizedVariants.txt file in the Unicode Character Database. Note that variation sequences are not defined for the bold script alphabet, nor for lowercase letters of the regular script alphabet, as there is no evidence of systematic distinctive use of variant styles for those ranges.

The Unicode code charts use the roundhand calligraphic style to display mathematical script letters, including the various script symbols encoded in the Letterlike Symbols block on the BMP. That choice is less disruptive for legacy fonts, and is more consistent with the expected display for the occasional use of such letterlike symbols in nonmathematical contexts such as the use of U+2133 SCRIPT CAPITAL M for the pre-1949 symbol for the German currency unit *Mark*. By contrast, widely deployed specialty mathematical fonts such as Cambria Math and STIX Two Math default to the chancery calligraphic style, which is the specific script variant currently favored by mathematicians.

Compatibility Decompositions. All mathematical alphanumeric symbols have compatibility decompositions to the base Latin and Greek letters. This does not imply that the use of these characters is discouraged for mathematical use. Folding away such distinctions by applying the compatibility mappings is usually not desirable, as it loses the semantic distinctions for which these characters were encoded. See Unicode Standard Annex #15, "Unicode Normalization Forms."

22.2.4 Fonts Used for Mathematical Alphabets

Mathematicians place strict requirements on the *specific* fonts used to represent mathematical variables. Readers of a mathematical text need to be able to distinguish single-letter variables from each other, even when they do not appear in close proximity. They must be able to recognize the letter itself, whether it is part of the text or is a mathematical variable, and lastly which mathematical alphabet it is from.

Fraktur. The blackletter style is often referred to as *Fraktur* or *Gothic* in various sources. Technically, Fraktur and Gothic typefaces are distinct designs from blackletter, but any of several font styles similar in appearance to the forms shown in the charts can be used. In East Asian typography, the term *Gothic* is commonly used to indicate a sans-serif type style.

Math Italics. Mathematical variables are most commonly set in a form of italics, but not all italic fonts can be used successfully. For example, a math italic font should avoid a "tail" on the lowercase *italic letter z* because it clashes with subscripts. In common text fonts, the *italic letter v* and *Greek letter nu* are not very distinct. A rounded *italic letter v* is therefore preferred in a mathematical font. There are other characters that sometimes have similar shapes and require special attention to avoid ambiguity. Examples are shown in *Figure 22-5*.

Figure 22-5. Easily Confused Shapes for Mathematical Glyphs

italic a	a	α	alpha
italic v (pointed)	v	ν	nu
italic v (rounded)	υ	υ	upsilon
script X	$\mathcal{X}$	χ	chi
plain Y	Y	Υ	Upsilon

Hard-to-Distinguish Letters. Not all sans-serif fonts allow an easy distinction between *lowercase l* and *uppercase I*, and not all monospaced (monowidth) fonts allow a distinction between the *letter l* and the *digit one*. Such fonts are not usable for mathematics. In Fraktur, the letters ℑ and ℨ, in particular, must be made distinguishable. Overburdened blackletter forms are inappropriate for mathematical notation.

Similarly, the *digit zero* must be distinct from the *uppercase letter O* for all mathematical alphanumeric sets. Some characters are so similar that even mathematical fonts do not attempt to provide distinct glyphs for them. Their use is normally avoided in mathematical notation unless no confusion is possible in a given context—for example, *uppercase A* and *uppercase Alpha*.

Font Support for Combining Diacritics. Mathematical equations require that characters be combined with diacritics (dots, tilde, circumflex, or arrows above are common), as well as followed or preceded by superscripted or subscripted letters or numbers. This requirement leads to designs for *italic* styles that are less inclined and *script* styles that have smaller overhangs and less slant than equivalent styles commonly used for text such as wedding invitations.

Double-Struck Characters. The double-struck glyphs shown in earlier editions of the standard attempted to match the design used for all the other Latin characters in the standard, which is based on Times. The current set of fonts was prepared in consultation with the American Mathematical Society and leading mathematical publishers; it shows much simpler forms that are derived from the forms written on a blackboard. However, both serifed and non-serifed forms can be used in mathematical texts, and inline fonts are found in works published by certain publishers.

22.2.5 Arabic Mathematical Alphabetic Symbols: U+1EE00– U+1EEFF

The Arabic Mathematical Alphabetic Symbols block contains a set of characters used to write Arabic mathematical expressions. These symbols derive from a version of the Arabic alphabet which was widely used for many centuries and in a variety of contexts, such as in manuscripts and traditional print editions. The characters in this block follow the older, generic Semitic order (a, b, j, d…), differing from the order typically found in dictionaries (a, b, t, th…). These symbols are used by Arabic alphabet-based scripts, such as Arabic and Persian, and appear in the majority of mathematical handbooks published in the Middle East, Libya, and Algeria today.

In Arabic mathematical notation, much as in Latin-based mathematical text, style variation plays an important semantic role and must be retained in plain text. Hence Arabic styles for these mathematical symbols, which include tailed, stretched, looped, or double-struck forms, are encoded separately, and should not be handled at the font level. These mathematically styled symbols, which also include some isolated and initial-form Arabic letters, are to be distinguished from the Arabic compatibility characters encoded in the Arabic Presentation Forms-B block.

Shaping. The Arabic Mathematical Symbols are not subject to shaping, unlike the Arabic letters in the Arabic block (U+0600..U+06FF).

Large Operators. Two operators are separately encoded: U+1EEF0 ARABIC MATHE-
MATICAL OPERATOR MEEM WITH HAH WITH TATWEEL, which denotes summation
in Arabic mathematics, and U+1EEF1 ARABIC MATHEMATICAL OPERATOR HAH
WITH DAL, which denotes limits in Persian mathematics. The glyphs for both of these
characters stretch, based on the width of the text above or below them.

Properties. The characters in this block, although used as mathematical symbols,
have the General_Category value Lo. This property assignment for these letterlike
symbols reflects the similar treatment for the alphanumeric mathematical symbols
based on Latin and Greek letterforms.

22.3 Numerals

Many characters in the Unicode Standard are used to represent numbers or numeric expressions. Some characters are used exclusively in a numeric context; other characters can be used both as letters and numerically, depending on context. The notational systems for numbers are equally varied. They range from the familiar decimal notation to non-decimal systems, such as Roman numerals.

Encoding Principles. The Unicode Standard encodes sets of digit characters (or non-digit characters, as appropriate) for each script which has significantly distinct forms for numerals. As in the case of encoding of letters (and other units) for writing systems, the emphasis is on encoding the units of the written forms for numeric systems.

Sets of digits which differ by mathematical style are separately encoded, for use in mathematics. Such mathematically styled digits may carry distinct semantics which is maintained as a plain text distinction in the representation of mathematical expressions. This treatment of styled digits for mathematics parallels the treatment of styled alphabets for mathematics. See "Mathematical Alphabets" in *Section 22.2, Letterlike Symbols*.

Other font face distinctions for digits which do not have mathematical significance, such as the use of old style digits in running text, are not separately encoded. Other glyphic variations in digits and numeric characters are likewise not separately encoded. There are a few documented exceptions to this general rule. See "Glyph Variants of Decimal Digits" later in this section.

22.3.1 Decimal Digits

A decimal digit is a digit that is used in decimal (radix 10) place value notation. The most widely used decimal digits are the European digits, encoded in the range from U+0030 DIGIT ZERO to U+0039 DIGIT NINE. Because of their early encoding history, these digits are also commonly known as *ASCII digits*. They are also known as *Western digits* or *Latin digits*. The European digits are used with a large variety of writing systems, including those whose own number systems are not decimal radix systems.

Many scripts also have their own decimal digits, which are separately encoded. Examples are the digits used with the Arabic script or those of the Indic scripts. *Table 22-3* lists scripts for which separate decimal digits are encoded, together with the section in the Unicode Standard which describes that script. The scripts marked with an asterisk (Arabic, Myanmar, and Tai Tham) have two or more sets of digits.

Table 22-3. Script-Specific Decimal Digits

Script	Section	Script	Section
Adlam	*Section 19.9*	Modi	*Section 15.12*
Ahom	*Section 15.16*	Mongolian	*Section 13.5*
Arabic*	*Section 9.2*	Mro	*Section 13.8*
Balinese	*Section 17.3*	Myanmar*	*Section 16.3*
Bengali & Assamese	*Section 12.2*	Nag Mundari	*Section 13.12*
Bhaiksuki	*Section 14.3*	New Tai Lue	*Section 16.6*
Brahmi	*Section 14.1*	Newa	*Section 13.3*
Chakma	*Section 13.13*	N'Ko	*Section 19.4*
Cham	*Section 16.10*	Nyiakeng Puachue Hmong	*Section 16.12*
Devanagari	*Section 12.1*	Ol Chiki	*Section 13.10*
Dives Akuru	*Section 15.15*	Oriya	*Section 12.5*
Garay	*Section 19.11*	Osmanya	*Section 19.2*
Gujarati	*Section 12.4*	Pahawh Hmong	*Section 16.11*
Gunjala Gondi	*Section 13.17*	Saurashtra	*Section 13.15*
Gurmukhi	*Section 12.3*	Sharada	*Section 15.3*
Gurung Khema	*Section 13.22*	Sinhala	*Section 13.2*
Hanifi Rohingya	*Section 16.14*	Sora Sompeng	*Section 15.17*
Javanese	*Section 17.4*	Sundanese	*Section 17.7*
Kannada	*Section 12.8*	Sunuwar	*Section 13.21*
Kawi	*Section 17.9*	Tai Tham*	*Section 16.7*
Kayah Li	*Section 16.9*	Takri	*Section 15.4*
Khmer	*Section 16.4*	Tamil	*Section 12.6*
Khudawadi	*Section 15.9*	Tangsa	*Section 13.20*
Kirat Rai	*Section 13.23*	Telugu	*Section 12.7*
Lao	*Section 16.2*	Thai	*Section 16.1*
Lepcha	*Section 13.14*	Tibetan	*Section 13.4*
Limbu	*Section 13.6*	Tirhuta	*Section 15.11*
Malayalam	*Section 12.9*	Vai	*Section 19.5*
Masaram Gondi	*Section 13.16*	Wancho	*Section 13.18*
Meetei Mayek	*Section 13.7*	Warang Citi	*Section 13.9*

In the Unicode Standard, a character is formally classified as a decimal digit if it meets the conditions set out in "Decimal Digits" in *Section 4.6, Numeric Value* and has been assigned the property Numeric_Type = Decimal. The Numeric_Type property can be used to get the complete list of all decimal digits for any version of the Unicode Standard. (See DerivedNumericType.txt in the Unicode Character Database.)

When characters classified as decimal digits are used in sequences to represent decimal radix numerals, they are always stored most significant digit first. This convention includes decimal digits associated with scripts whose predominant layout direction is right-to-left. The visual layout of decimal radix numerals in bidirectional contexts depends on the interaction of their Bidi_Class values with the Unicode Bidirectional Algorithm (UBA). In many cases, decimal digits share the same strong Bidi_Class values with the letters of their script ("L" or "R"). A few common-use decimal digits, such as the ASCII digits and the Arabic script digits have special Bidi_Class values that interact with dedicated rules for resolving the direction of numbers in the UBA. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm.")

The Unicode Standard does not specify which sets of decimal digits can or should be used with any particular writing system, language, or locale. However, the information provided in the Unicode Common Locale Data Repository (CLDR) contains information about which set or sets of digits are used with particular locales defined in CLDR. Numeral systems for a given locale require additional information, such as the appropriate decimal and grouping separators, the type of digit grouping used, and so on; that information is also supplied in CLDR.

Exceptions. There are several scripts with exceptional encodings for characters that are used as decimal digits. For the Arabic script, there are two sets of decimal digits encoded which have somewhat different glyphs and different directional properties. See "Arabic-Indic Digits" in *Section 9.2, Arabic* for a discussion of these two sets and their use in Arabic text. For the Myanmar script a second set of digits is encoded for the Shan language, and a third set of digits is encoded for the Tai Laing language. The Tai Tham script also has two sets of digits, which are used in different contexts.

CJK Ideographs Used as Decimal Digits. The CJK ideographs listed in *Table 4-5*, with numeric values in the range one through nine, can be used in decimal notations (with 0 represented by U+3007 IDEOGRAPHIC NUMBER ZERO). These ideographic digits are not coded in a contiguous sequence, nor do they occur in numeric order. Unlike other script-specific digits, they are not uniquely used as decimal digits. The same characters may be used in the traditional Chinese system for writing numbers, which is not a decimal radix system, but which instead uses numeric symbols for tens, hundreds, thousands, ten thousands, and so forth. See *Figure 22-6*, which illustrates two different ways the number 1,234 can be written with CJK ideographs.

Figure 22-6. CJK Ideographic Numbers

一千二百三十四

or

一二三四

CJK numeric ideographs are also used in word compounds which are not interpreted as numbers. Parsing CJK ideographs as decimal numbers therefore requires information about the context of their use.

22.3.2 Other Digits

Hexadecimal Digits. Conventionally, the letters "A" through "F", or their lowercase equivalents are used with the ASCII decimal digits to form a set of hexadecimal digits. These characters have been assigned the Hex_Digit property. Although overlapping the letters and digits this way is not ideal from the point of view of numerical parsing, the practice is long standing; nothing would be gained by encoding a new, parallel, separate set of hexadecimal digits.

Compatibility Digits. There are a several sets of compatibility digits in the Unicode Standard. *Table 22-4* provides a full list of compatibility digits.

Table 22-4. Compatibility Digits

Description	Code Range(s)	Numeric Type	Decomp Type	Section
Fullwidth digits	FF10..FF19	Decimal	Wide	*Section 18.5*
Bold digits	1D7CE..1D7D7	Decimal	Font	*Section 22.2*
Double struck	1D7D8..1D7E1	Decimal	Font	*Section 22.2*
Monospace digits	1D7F6..1D7FF	Decimal	Font	*Section 22.2*
Sans-serif digits	1D7E2..1D7EB	Decimal	Font	*Section 22.2*
Sans-serif bold digits	1D7EC..1D7F5	Decimal	Font	*Section 22.2*
Segmented digits	1FBF0..1FBF9	Decimal	Font	*Section 22.7*
Superscript digits	2070, 00B9, 00B2, 00B3, 2074..2079	Digit	Super	*Section 22.4*
Subscript digits	2080..2089	Digit	Sub	*Section 22.4*
Circled digits	24EA, 2460..2468	Digit	Circle	
Parenthesized digits	2474..247C	Digit	Compat	
Digits plus full stop	1F100, 2488..2490	Digit	Compat	
Digits plus comma	1F101..1F10A	Digit	Compat	
Double circled digits	24F5..24FD	Digit	None	
Dingbat negative circled digits	2776..277E	Digit	None	
Dingbat circled sans-serif digits	1F10B, 2780..2788	Numeric or Digit	None	
Dingbat negative circled sans-serif digits	1F10C, 278A..2792	Numeric or Digit	None	
Segmented Digits	1FBF0..1FBF9	Decimal	Font	*Section 22.7*

The fullwidth digits are simply wide presentation forms of ASCII digits, occurring in East Asian typographical contexts. They have compatibility decompositions to ASCII digits, have Numeric_Type = Decimal, and should be processed as regular decimal digits.

The various mathematically styled digits in the range U+1D7CE..U+1D7F5 are specifically intended for mathematical use. They also have compatibility decompositions to ASCII digits and meet the criteria for Numeric_Type = Decimal. Although they may have particular mathematical meanings attached to them, in most cases it would be safe for generic parsers to simply treat them as additional sets of decimal digits.

The segmented digits encoded in the range U+1FBF0..U+1FBF9 are used in legacy terminal applications, and are essentially just another styled set of ASCII digits. It is also safe for generic parsers to treat them as an additional set of decimal digits.

Parsing of Superscript and Subscript Digits. In the Unicode Character Database, superscript and subscript digits have not been given the General_Category property value Decimal_Number (gc = Nd); correspondingly, they have the Numeric_Type property value Digit, rather than Decimal. This is to prevent superscripted expressions like 2^3 from being interpreted as 23 by simplistic parsers. More sophisticated numeric parsers, such as general mathematical expression parsers, should correctly identify these compatibility superscript and subscript characters as digits and interpret them appropriately. Note that the compatibility superscript digits are not encoded in a single, contiguous range.

For mathematical notation, the use of superscript or subscript styling of ASCII digits is preferred over the use of compatibility superscript or subscript digits. See Unicode Technical Report #25, "Unicode Support for Mathematics," for more discussion of this topic.

Numeric Bullets. The other sets of compatibility digits listed in *Table 22-4* are typically derived from East Asian legacy character sets, where their most common use is as numbered text bullets. Most occur as part of sets which extend beyond the value 9 up to 10, 12, or even 50. Most are also defective as sets of digits because they lack a value for 0. None is given the Numeric_Type of Decimal. Only the basic set of simple circled digits is given compatibility decompositions to ASCII digits. The rest either have compatibility decompositions to digits plus punctuation marks or have no decompositions at all. Effectively, all of these numeric bullets should be treated as dingbat symbols with numbers printed on them; they should not be parsed as representations of numerals.

Glyph Variants of Decimal Digits. Some variations of decimal digits are considered glyph variants and are not separately encoded. These include the old style variants of digits, as shown in *Figure 22-7*. Glyph variants of the digit zero with a centered dot or

a diagonal slash to distinguish it from the uppercase letter "O", or of the digit seven with a horizontal bar to distinguish it from handwritten forms for the digit one, are likewise not separately encoded.

Figure 22-7. Regular and Old Style Digits

Regular Digits: 0 1 2 3 4 5 6 7 8 9

Old Style Digits: 0 1 2 3 4 5 6 7 8 9

In a few cases, such as for a small number of mathematical symbols, there may be a strong rationale for the unambiguous representation of a certain glyph variant of a decimal digit. In particular, the glyph variant of the digit zero with a short diagonal stroke "0" can be unambiguously represented with the standardized variation sequence <U+0030, U+FE00>.

Significant regional glyph variants for the Eastern-Arabic Digits U+06F0..U+06F9 also occur, but are not separately encoded. See *Table 9-2* for illustrations of those variants.

Accounting Numbers. Accounting numbers are variant forms of digits or other numbers designed to deter fraud. They are used in accounting systems or on various financial instruments such as checks. These numbers often take shapes which cannot be confused with other digits or letters, and which are difficult to convert into another digit or number by adding on to the written form. When such numbers are clearly distinct characters, as opposed to merely glyph variants, they are separately encoded in the standard. The use of accounting numbers is particularly widespread in Chinese and Japanese, because the Han ideographs for one, two, and three have simple shapes that are easy to convert into other numbers by forgery. See *Table 4-6*, for a list of the most common alternate ideographs used as accounting numbers for the traditional Chinese numbering system.

Characters for accounting numbers are occasionally encoded separately for other scripts as well. For example, U+19DA NEW TAI LUE THAM DIGIT ONE is an accounting form for the digit one which cannot be confused with the vowel sign *-aa* and which cannot easily be converted into the digit for three.

22.3.3 Non-Decimal Radix Systems

A number of scripts have number systems that are not decimal place-value notations. Such systems are fairly common among traditional writing systems of South Asia. The following provides descriptions or references to descriptions of non-decimal radix systems elsewhere in the Standard.

Ethiopic Numerals. The Ethiopic script contains digits and other numbers for a traditional number system which is not a decimal place-value notation. This traditional system does not use a zero. It is further described in <u>*Section 19.1, Ethiopic*</u>.

Mende Kikakui Numerals. The Mende Kikakui script has a unique set of numerals, constituting a set of digits one through nine, used with a set of multiplier subscripts for powers of ten from 10 through 1,000,000. For more details on the structure of this numeral system, including examples, see <u>*Section 19.8, Mende Kikakui*</u>.

Medefaidrin Numerals. The numerals used with the Medefaidrin script (see <u>*Section 19.10, Medefaidrin*</u>) constitute a novel, vigesimal radix system, with "digits" in the range 0 to 19. The Medefaidrin script is used only by a small community for religious purposes, so little is known about the practical use of these numerals.

Mayan Numerals. Mayan writing used a set of vigesimal numerals, including a sign for zero. These signs are very well-known from Mayan calendrical inscriptions. They are striking in form, consisting of a series of horizontal bars with varying numbers of large dots above the bars, and so are easy to spot in inscriptions, amidst all the other hieroglyphic signs based on heads, animals, and so forth. The Mayan numerals are so well known, in fact, that they have gained a degree of modern re-use, appearing, for example, in page numbering of small documents published in Guatemala or the Yucatan. To accommodate such modern use of Mayan numerals, the full set has been encoded in the range U+1D2E0..U+1D2F3 in a dedicated Mayan Numerals block.

Until the analysis and encoding of the complex Mayan hieroglyphic script can be completed, these Mayan numerals stand by themselves. They are not given a Mayan Script property value, but are instead just treated as numeric symbols with the Script property Common.

Kaktovik Numerals. Kaktovik numerals are vigesimal numerals including a sign for zero. These signs were devised by speakers of Iñupiaq in Kaktovik, Alaska for the counting systems of Inuit and Yupik languages. The top part of each numeral consists of up to three horizontal strokes, marking the fives, while the bottom part consists of up to four vertical strokes, marking the ones. The strokes are joined into a single continuous line.

Because the system is positional, for example U+1D2C1 KAKTOVIK NUMERAL ONE can indicate 1, 20, 400, 8,000, and so on, and U+1D2C5 KAKTOVIK NUMERAL FIVE indicates five times those amounts. Each Kaktovik numeral from 0 through 19 is encoded atomically.

Cuneiform Numerals. Sumero-Akkadian numerals were used for sexagesimal systems. There was no symbol for zero, but by Babylonian times, a place value system was in use. Thus the exact value of a digit depended on its position in a number. There was also ambiguity in numerical representation, because a symbol such as U+12079 CUNEIFORM SIGN DISH could represent either 1 or 1×60 or $1 \times (60 \times 60)$,

depending on the context. A numerical expression might also be interpreted as a sexagesimal fraction. So the sequence <1, 10, 5> might be evaluated as $1 \times 60 + 10 + 5 = 75$ or $1 \times 60 \times 60 + 10 + 5 = 3615$ or $1 + (10 + 5)/60 = 1.25$. Many other complications arise in Cuneiform numeral systems, and they clearly require special processing distinct from that used for modern decimal radix systems. For more information, see *Section 11.1, Sumero-Akkadian*.

Other Ancient Numeral Systems. A number of other ancient numeral systems have characters encoded for them. Many of these ancient systems are variations on tallying systems. In numerous cases, the data regarding ancient systems and their use is incomplete, because of the fragmentary nature of the ancient text corpuses. Characters for numbers are encoded, however, to enable complete representation of the text which does exist.

Ancient Aegean numbers were used with the Linear A and Linear B scripts, as well as the Cypriot syllabary. They are described in *Section 8.2, Linear B*.

Many of the ancient Semitic scripts had very similar numeral systems which used tally-shaped numbers for one, two, and three, and which then grouped those, along with some signs for tens and hundreds, to form larger numbers. See the discussion of these systems in *Section 10.3, Phoenician* and, in particular, the discussion with examples of number formation in *Section 10.4, Imperial Aramaic*.

22.3.4 Acrophonic Systems and Other Letter-based Numbers

There are many instances of numeral systems, particularly historic ones, which use letters to stand for numbers. In some cases these systems may coexist with numeral systems using separate digits or other numbers. Two important sub-types are acrophonic systems, which assign numeric values based on the letters used for the initial sounds of number words, and alphabetic numerals, which assign numeric values based roughly on alphabetic order. A well-known example of a partially acrophonic system is the Roman numerals, which include c(entum) and m(ille) for 100 and 1000, respectively. The Greek Milesian numerals are an example of an alphabetic system, with alpha = 1, beta = 2, gamma = 3, and so forth.

In the Unicode Standard, although many letters in common scripts are known to be used for such letter-based numbers, they are not given numeric properties unless their *only* use is as an extension of an alphabet specifically for numbering. In most cases, the interpretation of letters or strings of letters as having numeric values is outside the scope of the standard.

Roman Numerals. For most purposes, it is preferable to compose the Roman numerals from sequences of the appropriate Latin letters. However, the uppercase and lowercase variants of the Roman numerals through 12, plus L, C, D, and M, have been encoded in the Number Forms block (U+2150..U+218F) for compatibility with

East Asian standards. Unlike sequences of Latin letters, these symbols remain upright in vertical layout. Additionally, in certain locales, compact date formats use Roman numerals for the month, but may expect the use of a single character.

In identifiers, the use of Roman numeral symbols—particularly those based on a single letter of the Latin alphabet—can lead to spoofing. For more information, see Unicode Technical Report #36, "Unicode Security Considerations."

U+2180 ROMAN NUMERAL ONE THOUSAND C D and U+216F ROMAN NUMERAL ONE THOUSAND can be considered to be glyphic variants of the same Roman numeral, but are distinguished because they are not generally interchangeable and because U+2180 cannot be considered to be a compatibility equivalent to the Latin letter M. U+2181 ROMAN NUMERAL FIVE THOUSAND and U+2182 ROMAN NUMERAL TEN THOUSAND are distinct characters used in Roman numerals; they do not have compatibility decompositions in the Unicode Standard. U+2183 ROMAN NUMERAL REVERSED ONE HUNDRED is a form used in combinations with C and/or I to form large numbers—some of which vary with single character number forms such as D, M, U+2181, or others. U+2183 is also used for the Claudian letter *antisigma*.

Greek Numerals. The ancient Greeks used a set of acrophonic numerals, also known as Attic numerals. These are represented in the Unicode Standard using capital Greek letters. A number of extensions for the Greek acrophonic numerals, which combine letterforms in odd ways, or which represent local regional variants, are separately encoded in the Ancient Greek Numbers block, U+10140..U+1018A.

Greek also has an alphabetic numeral system, called Milesian or Alexandrian numerals. These use the first third of the Greek alphabet to represent 1 through 9, the middle third for 10 through 90, and the last third for 100 through 900. U+0374 GREEK NUMERAL SIGN (the *dexia keraia*) marks letters as having numeric values in modern typography. U+0375 GREEK LOWER NUMERAL SIGN (the *aristeri keraia*) is placed on the left side of a letter to indicate a value in the thousands.

In Byzantine and other Greek manuscript traditions, numbers were often indicated by a horizontal line drawn above the letters being used as numbers. The Coptic script uses similar conventions. See *Section 7.3, Coptic*.

22.3.5 Coptic Epact Numbers: U+102E0–U+102FF

The Coptic epact numbers are elements of a decimal sign-value notation system used in some Coptic manuscripts. These numbers are referred to as "epact," based on the Greek word ἐπακτός "imported." They differ from the usual representation of numbers in Coptic texts, which consists of a system assigning numeric values directly to letters of the Coptic alphabet.

The Coptic epact numbers are considered to be historically derived from cursive forms of ordinary Coptic letters. They were developed in the 10th century CE by the Coptic community for administrative purposes. They are primarily attested in Coptic manuscripts written in Arabic, such as astronomical texts. They also appear in some accounting documents.

The numerical system for Coptic epact numbers is additive. The value of a numeric sequence consists of the sum of each number in the sequence. There is no character for zero. Instead, there are three sets of signs for the values 1 through 9, representing three orders: the digits, the tens, and the hundreds.

Numeric sequences are written from left to right, starting with the largest number at the left. For example, 25 is written ⲱⲉ <U+102EB twenty, U+102E5 five>; 205 is written ⲥⲉ <U+102F4 two hundred, U+102E5 five>; 250 is written ⲥⲩ <U+102F4 two hundred, U+102EE fifty>. This order is followed even when Coptic epact numbers are embedded in right-to-left Arabic text.

Larger numbers are represented by applying a sublinear diacritical mark, U+102E0 COPTIC EPACT THOUSANDS MARK. Essentially, this mark multiplies the value of its base character by one thousand. Thus, when applied to symbols from the digits order, it represents thousands; when applied to symbols from the tens order, it represents ten thousands, and so on. A second application of the sublinear diacritic multiplies the base value by another factor of one thousand.

Ordinary Coptic numbers are often distinguished from Coptic letters by marking them with a line above. (See *Section 7.3, Coptic*.) A visually similar convention is also seen for Coptic epact numbers, where an entire numeric sequence may be marked with a wavy line above. This mark is represented by U+0605 ARABIC NUMBER MARK ABOVE. As when used with Arabic digits, ARABIC NUMBER MARK ABOVE *precedes* the sequence of Coptic epact numbers in the underlying representation, and is rendered across the top of the entire sequence for display.

22.3.6 Rumi Numeral Symbols: U+10E60–U+10E7F

Rumi, also known today as Fasi, is an numeric system used from the 10th to 17th centuries CE in a wide area, spanning from Egypt, across the Maghreb, to al-Andalus on the Iberian Peninsula. The Rumi numerals originate from the Coptic or Greek-Coptic tradition, but are not a positionally-based numbering system.

The numbers appear in foliation, chapter, and quire notations in manuscripts of religious, scientific, accounting and mathematical works. They also were used on astronomical instruments.

There is considerable variety in the Rumi glyph shapes over time: the digit "nine," for example, appears in a theta shape in the early period. The glyphs in the code charts

derive from a copy of a manuscript by Ibn Al-Banna (1256–1321), with glyphs that are similar to those found in 16th century manuscripts from the Maghreb.

22.3.7 Siyaq Numerical Notation Systems

There are a number of regional traditions for numerical notation systems known as Siyaq, derived from the Arabic word *siyāq*, meaning "order." These traditions consist of specialized subsets of the Arabic script, formerly used in accounting and for general recording of numbers. A notable feature of Siyaq traditions is the use of stylized monograms of the Arabic names for numbers, rather than the ordinary Arabic-Indic digits.

Siyaq numbers represent units of a decimal positional system. The systems are additive—that is, the numeric value of a complete Siyaq number sequence consists of the sum of all the characters. There is no character for zero; instead, zero is represented inherently in the distinct numbers for the various decimal orders. Typically, there are distinctive numbers for the primary units, tens, hundreds, thousands, and ten thousands. The hundred thousands, millions, and higher orders are represented using unit marks and numbers from the smaller orders.

Siyaq numbers are written from right to left in the regular manner of the Arabic script. This orientation differs from the Arabic-Indic digits, which are written from left to right. In a Siyaq sequence, the largest number occurs first, and smaller units follow, laid out in visual order toward the left. An exception occurs for compound numbers of the tens and primary units; these are written transposed, with a "prefixed" form of the primary unit placed before the larger number.

Ottoman Siyaq. The Ottoman, or Turkish, Siyaq numbers are encoded in the Ottoman Siyaq Numbers block (U+1ED00..U+1ED4F). These are also known as *Siyakat* numbers. The system contains several alternate forms for numbers, which may be historical retentions. These alternate forms are encoded as distinct characters for the numbers two through ten and for a few other numbers of higher orders.

The Ottoman Siyaq system includes a specialized multiplier character, U+1ED2E OTTOMAN SIYAQ MARRATAN (from the Arabic word *marratan*, "multiplier"). The multiplier is used in combination with *one hundred* and *one thousand* for expressing the millions and larger orders.

Ottoman Siyaq also uses a number of fractions. These fractions may be written in sequence after the number, or may be rendered beneath the number. Because of their distinctive shapes, two of the fractions are encoded as separate numeric symbols: U+1ED3C OTTOMAN SIYAQ FRACTION ONE HALF and U+1ED3D OTTOMAN SIYAQ FRACTION ONE SIXTH.

In some Ottoman Siyaq sources, a baseline dot indicates the end of a numerical sequence, and is placed after the last number. The dot can be represented either by U+002E FULL STOP or U+06D4 ARABIC FULL STOP, depending on the desired shape of the numerical terminator.

Indic Siyaq. The Indic Siyaq tradition is known in India and other parts of South Asia as *raqm* or *rakam*, from the Arabic word *raqm*, meaning "account." Indic Siyaq is encoded in the Indic Siyaq Numbers block (U+1EC70..U+1ECBF). Like other Siyaq traditions, Indic Siyaq uses stylized monograms of the Arabic names for numbers, but the numbers for large decimal orders are derived from words of Indic languages. The period during which Siyaq was introduced in India is difficult to determine. The system was in common use under the Mughals by the 17th century, and remained in use into the middle of the 20th century.

There are two major styles of Siyaq used in India: the northern style and the "Deccani" or southern style. In general, the number forms and notation system of the two are identical. Minor points of difference lie in the orthography for the thousands, ten thousands, and lakhs.

There are also some minor style variations in writing tens of *lakhs* (millions) and tens of *crores* (hundred millions). For example, when writing the number ten *lakh*, one style may use a looped form of ten, looking like U+1EC7A INDIC SIYAQ NUMBER TEN, but another may use a straight form, looking like U+1EC95 INDIC SIYAQ NUMBER TEN THOUSAND. Such differences in style should be considered orthographic differences. The visual form seen in the documents being represented should be used to represent Indic Siyaq text. Thus, in one style the number ten *lakh* (one million) would be represented as <U+1EC7A, U+1ECA0>, but in another style as <U+1EC95, U+1ECA0>. Processes that interpret Indic Siyaq numbers should be aware of this irregular use of tens of thousands (U+1EC95..U+1EC9D) for tens when they appear before *lakhs* and *crores*.

The Indic Siyaq numbers are generally used within an Arabic script environment and within Urdu and Persian linguistic contexts. They may also occur in multilingual environments alongside other scripts. Arabic-Indic digits occasionally occur within Siyaq sequences, particularly for the representation of small currency units.

22.3.8 CJK Numerals

CJK Ideographic Traditional Numerals. The traditional Chinese system for writing numerals is not a decimal radix system. It is decimal-based, but uses a series of decimal counter symbols that function somewhat like tallies. So for example, the representation of the number 12,346 in the traditional system would be by a sequence of CJK ideographs with numeric values as follows: . See *Table 4-5* for a list of all the CJK ideographs for digits and decimal counters used in this system. The traditional system is still in widespread use, not only in China and other countries where

Chinese is used, but also in countries whose writing adopted Chinese characters—most notably, in Japan. In both China and Japan the traditional system now coexists with very common use of the European digits.

Chinese Counting-Rod Numerals. Counting-rod numerals were used in pre-modern East Asian mathematical texts in conjunction with counting rods used to represent and manipulate numbers. The counting rods were a set of small sticks, several centimeters long that were arranged in patterns on a gridded counting board. Counting rods and the counting board provided a flexible system for mathematicians to manipulate numbers, allowing for considerable sophistication in mathematics.

The specifics of the patterns used to represent various numbers using counting rods varied, but there are two main constants: Two sets of numbers were used for alternate columns; one set was used for the ones, hundreds, and ten-thousands columns in the grid, while the other set was used for the tens and thousands. The shapes used for the counting-rod numerals in the Unicode Standard follow conventions from the Song dynasty in China, when traditional Chinese mathematics had reached its peak. Fragmentary material from many early Han dynasty texts shows different orientation conventions for the numerals, with horizontal and vertical marks swapped for the digits and tens places.

Zero was indicated by a blank square on the counting board and was either avoided in written texts or was represented with U+3007 IDEOGRAPHIC NUMBER ZERO. (Historically, U+3007 IDEOGRAPHIC NUMBER ZERO originated as a dot; as time passed, it increased in size until it became the same size as an ideograph. The actual size of U+3007 IDEOGRAPHIC NUMBER ZERO in mathematical texts varies, but this variation should be considered a font difference.) Written texts could also take advantage of the alternating shapes for the numerals to avoid having to explicitly represent zero. Thus 6,708 can be distinguished from 678, because the former would be ⊥ 𝍠 𝍤, whereas the latter would be 𝍡 ⊥ 𝍤.

Negative numbers were originally indicated on the counting board by using rods of a different color. In written texts, a diagonal slash from lower right to upper left is overlaid upon the rightmost digit. On occasion, the slash might not be actually overlaid. U+20E5 COMBINING REVERSE SOLIDUS OVERLAY should be used for this negative sign.

The predominant use of counting-rod numerals in texts was as part of diagrams of counting boards. They are, however, occasionally used in other contexts, and they may even occur within the body of modern texts.

Suzhou-Style Numerals. The Suzhou-style numerals are CJK ideographic number forms encoded in the CJK Symbols and Punctuation block in the ranges U+3021..U+3029 and U+3038..U+303A.

The Suzhou-style numerals are modified forms of CJK ideographic numerals that are used by shopkeepers in China to mark prices. They are also known as "commercial forms," "shop units," or "grass numbers." They are encoded for compatibility with the CNS 11643-1992 and Big Five standards. The forms for ten, twenty, and thirty, encoded at U+3038..U+303A, are also encoded as CJK unified ideographs: U+5341, U+5344, and U+5345, respectively. (For twenty, see also U+5EFE and U+5EFF.)

These commercial forms of Chinese numerals should be distinguished from the use of other CJK unified ideographs as accounting numbers to deter fraud. See *Table 4-6* in *Section 4.6, Numeric Value*, for a list of ideographs used as accounting numbers.

Why are the Suzhou numbers called Hangzhou numerals in the Unicode names? No one has been able to trace this back. Hangzhou is a district in China that is near the Suzhou district, but the name "Hangzhou" does not occur in other sources that discuss these number forms.

22.3.9 Fractions

The Number Forms block (U+2150..U+218F) contains a series of vulgar fraction characters, encoded for compatibility with legacy character encoding standards. These characters are intended to represent both of the common forms of vulgar fractions: forms with a right-slanted division slash, such as ¼, as shown in the code charts, and forms with a horizontal division line, such as ¼, which are considered to be alternative glyphs for the same fractions, as shown in *Figure 22-8*. A few other vulgar fraction characters are located in the Latin-1 block in the range U+00BC..U+00BE.

Figure 22-8. Alternate Forms of Vulgar Fractions

The unusual fraction character, U+2189 VULGAR FRACTION ZERO THIRDS, is in origin a baseball scoring symbol from the Japanese television standard, ARIB STD B24. For baseball scoring, this character and the related fractions, U+2153 VULGAR FRACTION ONE THIRD and U+2154 VULGAR FRACTION TWO THIRDS, use the glyph form with the slanted division slash, and do not use the alternate stacked glyph form.

The vulgar fraction characters are given compatibility decompositions using U+2044 "/" FRACTION SLASH. Use of the *fraction slash* is the more generic way to represent fractions in text; it can be used to construct fractional number forms that are not included in the collections of vulgar fraction characters. For more information on the *fraction slash*, see "Other Punctuation" in *Section 6.2, General Punctuation*.

22.3.10 Common Indic Number Forms: U+A830–U+A83F

The Common Indic Number Forms block contains characters widely used in traditional representations of fractional values in numerous scripts of North India, Pakistan and in some areas of Nepal. They are also regularly used in several scripts of South India, including Kannada. The fraction signs were used to write currency, weight, measure, time, and other units. Their use in written documents is attested from at least the 16th century CE and in texts printed as late as 1970. They are occasionally still used in a limited capacity.

The North Indic fraction signs represent fraction values of a base-16 notation system. There are atomic symbols for 1/16, 2/16, 3/16 and for 1/4, 2/4, and 3/4. Intermediate values such as 5/16 are written additively by using two of the atomic symbols: 5/16 = 1/4 + 1/16, and so on. Some regional variation is found in the exact shape of the fraction signs used. For example, in Kannada, the fraction signs in the U+A833..U+A835 range are displayed with horizontal bars, instead of bars slanting upward to the right.

The signs for the fractions 1/4, 1/2, and 3/4 sometimes take different forms when they are written independently, without a currency or quantity mark. These independent forms were used more generally in Maharashtra and Gujarat, and they appear in materials written and printed in the Devanagari and Gujarati scripts. The independent fraction signs are represented by using middle dots to the left and right of the regular fraction signs.

U+A836 NORTH INDIC QUARTER MARK is used in some regional orthographies to explicitly indicate fraction signs for 1/4, 1/2, and 3/4 in cases where sequences of other marks could be ambiguous in reading.

This block also contains several other symbols that are not strictly number forms. They are used in traditional representation of numeric amounts for currency, weights, and other measures in the North Indic orthographies which use the fraction signs. U+A837 NORTH INDIC PLACEHOLDER MARK is a symbol used in currency representations to indicate the absence of an intermediate value. U+A839 NORTH INDIC QUANTITY MARK is a unit mark for various weights and measures.

The North Indic fraction signs are related to fraction signs that have specific forms and are separately encoded in some North Indic scripts. See, for example, U+09F4 BENGALI CURRENCY NUMERATOR ONE. Similar forms are attested for the Oriya script.

22.4 Superscript and Subscript Symbols

In general, the Unicode Standard does not attempt to describe the positioning of a character above or below the baseline in typographical layout. Therefore, the preferred means to encode superscripted letters or digits, such as "1$^{\text{st}}$" or "DC00$_{16}$", is by style or markup in rich text. However, in some instances superscript or subscript letters are used as part of the plain text content of specialized phonetic alphabets, such as the Uralic Phonetic Alphabet. These superscript and subscript letters are mostly from the Latin or Greek scripts. These characters are encoded in other character blocks, along with other modifier letters or phonetic letters. In addition, superscript digits are used to indicate tone in transliteration of many languages. The use of *superscript two* and *superscript three* is common legacy practice when referring to units of area and volume in general texts.

22.4.1 Superscripts and Subscripts: U+2070–U+209F

A certain number of additional superscript and subscript characters are needed for round-trip conversions to other standards and legacy code pages. Most such characters are encoded in this block and are considered compatibility characters.

Parsing of Superscript and Subscript Digits. In the Unicode Character Database, superscript and subscript digits have not been given the General_Category property value Decimal_Number (gc = Nd), so as to prevent expressions like 2^3 from being interpreted like 23 by simplistic parsers. This should not be construed as preventing more sophisticated numeric parsers, such as general mathematical expression parsers, from correctly identifying these compatibility superscript and subscript characters as digits and interpreting them appropriately. See also the discussion of digits in *Section 22.3, Numerals*.

Standards. Many of the characters in the Superscripts and Subscripts block are from character sets registered in the ISO International Register of Coded Character Sets to be Used With Escape Sequences, under the registration standard ISO/IEC 2375, for use with ISO/IEC 2022. Two MARC 21 character sets used by libraries include the digits, plus signs, minus signs, and parentheses.

Superscripts and Subscripts in Other Blocks. The superscript digits one, two, and three are coded in the Latin-1 Supplement block to provide code point compatibility with ISO/IEC 8859-1. For a discussion of U+00AA FEMININE ORDINAL INDICATOR and U+00BA MASCULINE ORDINAL INDICATOR, see "Letters of the Latin-1 Supplement" in *Section 7.1, Latin*. U+2120 SERVICE MARK and U+2122 TRADE MARK SIGN are commonly used symbols that are encoded in the Letterlike Symbols block (U+2100..U+214F); they consist of sequences of two superscripted letters each.

For phonetic usage, there are a small number of superscript letters located in the Spacing Modifier Letters block (U+02B0..U+02FF) and a large number of superscript and subscript letters in the Phonetic Extensions block (U+1D00..U+1D7F) and in the Phonetic Extensions Supplement block (U+1D80..U+1DBF). Those superscript and subscript letters function as modifier letters. The subset of those characters that are superscripted contain the words "modifier letter" in their names, instead of "superscript." The two superscript Latin letters in the Superscripts and Subscripts block, U+2071 SUPERSCRIPT LATIN SMALL LETTER I and U+207F SUPERSCRIPT LATIN SMALL LETTER N are considered part of that set of modifier letters; the difference in the naming conventions for them is an historical artifact, and is not intended to convey a functional distinction in the use of those characters in the Unicode Standard.

There are also a number of superscript or subscript symbols encoded in the Spacing Modifier Letters block (U+02B0..U+02FF). These symbols also often have the words "modifier letter" in their names, but are distinguished from most modifier letters by having the General_Category property value Sk. Like most modifier letters, the usual function of these superscript or subscript symbols is to indicate particular modifications of sound values in phonetic transcriptional systems. Characters such as U+02C2 MODIFIER LETTER LEFT ARROWHEAD or U+02F1 MODIFIER LETTER LOW LEFT ARROWHEAD should not be used to represent normal mathematical relational symbols such as U+003C "<" LESS-THAN SIGN in superscripted or subscripted expressions.

Finally, a small set of superscripted CJK ideographs, used for the Japanese system of syntactic markup of Classical Chinese text for reading, is located in the Kanbun block (U+3190..U+319F).

22.5 Mathematical Symbols

The Unicode Standard provides a large set of standard mathematical characters to support publications of scientific, technical, and mathematical texts on and off the Web. In addition to the mathematical symbols and arrows contained in the blocks described in this section, mathematical operators are found in the Basic Latin (ASCII) and Latin-1 Supplement blocks. These include U+002B PLUS SIGN, U+00D7 MULTIPLICATION SIGN and U+00F7 DIVISION SIGN, as well as U+003C LESS-THAN SIGN, U+003D EQUALS SIGN and U+003E GREATER-THAN SIGN. The *factorial operator* is unified with U+0021 EXCLAMATION MARK.

A few of the symbols from the Miscellaneous Technical, Miscellaneous Symbols, and Dingbats blocks, as well as characters from General Punctuation, are also used in mathematical notation. For Latin and Greek letters in special font styles that are used as mathematical variables, such as U+210B ℋ SCRIPT CAPITAL H, as well as the Hebrew letter *alef* used as the first transfinite cardinal symbol encoded by U+2135 ℵ ALEF SYMBOL, see "Letterlike Symbols" and "Mathematical Alphanumeric Symbols" in *Section 22.2, Letterlike Symbols*.

The repertoire of mathematical symbols in Unicode enables the display of virtually all standard mathematical symbols. Nevertheless, no collection of mathematical symbols can ever be considered complete; mathematicians and other scientists are continually inventing new mathematical symbols. More symbols will be added as they become widely accepted in the scientific communities.

Semantics. The same mathematical symbol may have different meanings in different subdisciplines or different contexts. The Unicode Standard encodes only a single character for a single symbolic form. For example, the "+" symbol normally denotes addition in a mathematical context, but it might refer to concatenation in a computer science context dealing with strings, indicate incrementation, or have any number of other functions in given contexts. It is up to the application to distinguish such meanings according to the appropriate context. For some common mathematical symbols there are also local variations in usage. For example, in addition to its long history of use as punctuation mark, U+00F7 DIVISION SIGN is also used in certain cases to indicate negative numbers in several European countries. Where information is available about the usage (or usages) of particular symbols, it has been indicated in the character annotations in the code charts.

Mathematical Property. The mathematical (*math*) property is an informative property of characters that are used as operators in mathematical formulas. The mathematical property may be useful in identifying characters commonly used in mathematical text and formulas. However, a number of these characters have multiple usages and may occur with nonmathematical semantics. For example, U+002D HYPHEN-MINUS may also be used as a hyphen—and not as a mathematical minus

sign. Other characters, including some alphabetic, numeric, punctuation, spaces, arrows, and geometric shapes, are used in mathematical expressions as well, but are even more dependent on the context for their identification. A list of characters with the mathematical property is provided in the Unicode Character Database.

For a classification of mathematical characters by typographical behavior and mapping to ISO 9573-13 entity sets, see Unicode Technical Report #25, "Unicode Support for Mathematics."

22.5.1 Mathematical Operators: U+2200–U+22FF

The Mathematical Operators block includes character encodings for operators, relations, geometric symbols, and a few other symbols with special usages confined largely to mathematical contexts.

Standards. Many national standards' mathematical operators are covered by the characters encoded in this block. These standards include such special collections as ANSI Y10.20, ISO 6862, ISO 8879, and portions of the collection of the American Mathematical Society, as well as the original repertoire of T_EX.

Encoding Principles. Mathematical operators often have more than one meaning. Therefore the encoding of this block is intentionally rather shape-based, with numerous instances in which several semantic values can be attributed to the same Unicode code point. For example, U+2205 ∅ EMPTY SET may denote the mathematical concept of *empty set* or the linguistic concept of *null morpheme* or *phonological "zero."* Similarly, U+2218 ∘ RING OPERATOR may be the equivalent of *white small circle* or *composite function* or *apl jot*. The Unicode Standard does not attempt to distinguish all possible semantic values that may be applied to mathematical operators or relation symbols.

The Unicode Standard does include many characters that appear to be quite similar to one another, but that may well convey different meanings in a given context. Conversely, mathematical operators, and especially relation symbols, may appear in various standards, handbooks, and fonts with a large number of purely graphical variants. Where variants were recognizable as such from the sources, they were not encoded separately.

Sometimes, specific glyph forms are chosen by notational style or are needed for contrast with other notation in the same document. For example, the symbol U+2205 ∅ EMPTY SET can be found in its slashed zero-shaped glyph form "Ø" in documents typeset in TeX, using the command *emptyset*, or in contexts where it is contrasted with the semantically distinct digit zero.

For this and certain other well-established glyph variants of mathematical symbols, standardized variation sequences were added to the Unicode Standard. Thus, for

example, the standardized variation sequence <U+2205, U+FE00> can be used to represent the variant "Ø" of the empty set symbol. To avoid the misuse of that sequence for the glyph variant of the digit zero with a short diagonal stroke "0", the standardized variation sequence <U+0030, U+FE00> was separately specified for that digit glyph variant.

For relation symbols, the choice of a vertical or forward-slanting stroke typically indicating negation often seems to be an aesthetic one, but either slant might appear in a given context. However, a back-slanted stroke almost always has a distinct meaning compared to the forward-slanted stroke. See *Section 23.4, Variation Selectors*, for more information on some particular variants.

Unifications. Mathematical operators such as *implies* ⇒ and *if and only if* ↔ have been unified with the corresponding arrows (U+21D2 RIGHTWARDS DOUBLE ARROW and U+2194 LEFT RIGHT ARROW, respectively) in the Arrows block.

The operator U+2208 ELEMENT OF is occasionally rendered with a taller shape than shown in the code charts. Mathematical handbooks and standards consulted treat these characters as variants of the same glyph. U+220A SMALL ELEMENT OF is a distinctively small version of the *element of* that originates in mathematical pi fonts.

The operators U+226B MUCH GREATER-THAN and U+226A MUCH LESS-THAN are sometimes rendered in a nested shape. The nested shapes are encoded separately as U+2AA2 DOUBLE NESTED GREATER-THAN and U+2AA1 DOUBLE NESTED LESS-THAN.

A large class of unifications applies to variants of relation symbols involving negation. Variants involving vertical or slanted *negation slashes* and *negation slashes* of different lengths are not separately encoded. For example, U+2288 NEITHER A SUBSET OF NOR EQUAL TO is the archetype for several different glyph variants noted in various collections.

In two instances in this block, essentially stylistic variants are separately encoded: U+2265 GREATER-THAN OR EQUAL TO is distinguished from U+2267 GREATER-THAN OVER EQUAL TO; the same distinction applies to U+2264 LESS-THAN OR EQUAL TO and U+2266 LESS-THAN OVER EQUAL TO. Further instances of the encoding of such stylistic variants can be found in the supplemental blocks of mathematical operators. The primary reason for such duplication is for compatibility with existing standards.

Disunifications. A number of mathematical operators have been disunified from related or similar punctuation characters, as shown in *Table 22-5*.

Table 22-5. Mathematical Operators Disunified from Punctuation

Punctuation	Mathematical Operator
002D - HYPHEN-MINUS	2212 − MINUS SIGN

002F / SOLIDUS or *slash*	2215 ∕ DIVISION SLASH
005C \ REVERSE SOLIDUS or *backslash*	2216 ∖ SET MINUS
002A * ASTERISK	2217 ∗ ASTERISK OPERATOR
25E6 ◦ WHITE BULLET	2218 ∘ RING OPERATOR
2022 • BULLET	2219 ∙ BULLET OPERATOR
007C | VERTICAL LINE	2223 ∣ DIVIDES
2016 ‖ DOUBLE VERTICAL LINE	2225 ∥ PARALLEL TO
003A : COLON	2236 ∶ RATIO
007E ~ TILDE	223C ∼ TILDE OPERATOR
00B7 · MIDDLE DOT	22C5 ⋅ DOT OPERATOR

These disunifications support specific mathematical semantics, as well as some significant display differences between the punctuation marks and the operators. Mathematical operators render on the math centerline, rather than the text baseline. Additionally, the angle or length of the operator counterparts of certain slashes or bars may differ from the corresponding punctuation marks. For certain pairs, such as COLON and RATIO, there are distinctions in the behavior of inter-character spacing; RATIO is rendered as a relational operator which takes visible space on both sides, whereas the punctuation mark COLON does not require such additional space in rendering.

The distinction between MIDDLE DOT and DOT OPERATOR deserves special consideration. DOT OPERATOR is preferred for mathematical use, where it signifies multiplication. This allows for rendering consistent with other mathematical operators, with unambiguous character properties and mathematical semantics. MIDDLE DOT is a legacy punctuation mark, with multiple uses, and with quite variable layout in different fonts. For the typographical convention of a *raised decimal point*, in contexts where simple layout is the priority and where automated parsing of decimal expressions is not required, MIDDLE DOT is the preferred representation.

In cases where there ordinarily is no rendering distinction between a punctuation mark and its use in mathematics, such as for U+0021 ! EXCLAMATION MARK used for *factorial* or for U+002E FULL STOP used for a *baseline decimal point*, there is no disunification, and only a single character has been encoded.

Greek-Derived Symbols. Several mathematical operators derived from Greek characters have been given separate encodings because they are used differently from the corresponding letters. These operators may occasionally occur in context with Greek-letter variables. They include U+2206 ∆ INCREMENT, U+220F ∏ N-ARY PRODUCT, and U+2211 ∑ N-ARY SUMMATION. The latter two are large operators that take limits.

Other duplicated Greek characters are those for U+00B5 µ MICRO SIGN in the Latin-1 Supplement block, U+2126 Ω OHM SIGN in Letterlike Symbols, and several characters among the APL functional symbols in the Miscellaneous Technical block. Most other

Greek characters with special mathematical semantics are found in the Greek block because duplicates were not required for compatibility. Additional sets of mathematical-style Greek alphabets are found in the Mathematical Alphanumeric Symbols block.

N-ary Operators. N-ary operators are distinguished from binary operators by their larger size and by the fact that in mathematical layout, they take limit expressions.

Invisible Operators. In mathematics, some operators or punctuation are often implied but not displayed. For a set of invisible operators that can be used to mark these implied operators in the text, see *Section 22.6, Invisible Mathematical Operators*.

Minus Sign. U+2212 "−" MINUS SIGN is a mathematical operator, to be distinguished from the ASCII-derived U+002D "-" HYPHEN-MINUS, which may look the same as a minus sign or be shorter in length. (For a complete list of dashes in the Unicode Standard, see *Table 6-3*.) U+22EE..U+22F1 are a set of ellipses used in matrix notation. U+2052 "⁒" COMMERCIAL MINUS SIGN is a specialized form of the minus sign. Its use is described in *Section 6.2, General Punctuation*.

Delimiters. Many mathematical delimiters are unified with punctuation characters. See *Section 6.2, General Punctuation*, for more information. Some of the set of ornamental brackets in the range U+2768..U+2775 are also used as mathematical delimiters. See *Section 22.9, Miscellaneous Symbols*. See also *Section 22.7, Technical Symbols*, for specialized characters used for large vertical or horizontal delimiters.

Bidirectional Layout. In a bidirectional context, with the exception of arrows, the glyphs for mathematical operators and delimiters are adjusted as described in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm." See *Section 4.7, Bidi Mirrored*, and "Paired Punctuation" in *Section 6.2, General Punctuation*.

Other Elements of Mathematical Notation. In addition to the symbols in these blocks, mathematical and scientific notation makes frequent use of arrows, punctuation characters, letterlike symbols, geometrical shapes, and miscellaneous and technical symbols.

For an extensive discussion of mathematical alphanumeric symbols, see *Section 22.2, Letterlike Symbols*. For additional information on all the mathematical operators and other symbols, see Unicode Technical Report #25, "Unicode Support for Mathematics."

22.5.2 Supplements to Mathematical Symbols and Arrows

The Unicode Standard defines a number of additional blocks to supplement the repertoire of mathematical operators and arrows. These additions are intended to extend the Unicode repertoire sufficiently to cover the needs of such applications as

MathML, modern mathematical formula editing and presentation software, and symbolic algebra systems.

Standards. MathML, an XML application, is intended to support the full legacy collection of the ISO mathematical entity sets. Accordingly, the repertoire of mathematical symbols for the Unicode Standard has been supplemented by the full list of mathematical entity sets in ISO TR 9573-13, *Public entity sets for mathematics and science*. An additional repertoire was provided from the amalgamated collection of the STIX Project (Scientific and Technical Information Exchange). That collection includes, but is not limited to, symbols gleaned from mathematical publications by experts of the American Mathematical Society and symbol sets provided by Elsevier Publishing and by the American Physical Society.

22.5.3 Supplemental Mathematical Operators: U+2A00–U+2AFF

The Supplemental Mathematical Operators block contains many additional symbols to supplement the collection of mathematical operators.

22.5.4 Miscellaneous Mathematical Symbols-A: U+27C0–U+27EF

The Miscellaneous Mathematical Symbols-A block contains symbols that are used mostly as operators or delimiters in mathematical notation.

Mathematical Brackets. The mathematical white square brackets, angle brackets, double angle brackets, and tortoise shell brackets encoded at U+27E6..U+27ED are intended for ordinary mathematical use of these particular bracket types. They are unambiguously narrow, for use in mathematical and scientific notation, and should be distinguished from the corresponding wide forms of white square brackets, angle brackets, and double angle brackets used in CJK typography. (See the discussion of the CJK Symbols and Punctuation block in *Section 6.2, General Punctuation*.) Note especially that the "bra" and "ket" angle brackets (U+2329 LEFT-POINTING ANGLE BRACKET and U+232A RIGHT-POINTING ANGLE BRACKET, respectively) are deprecated. Their use is strongly discouraged, because of their canonical equivalence to CJK angle brackets. This canonical equivalence is likely to result in unintended spacing problems if these characters are used in mathematical formulae.

The flattened parentheses encoded at U+27EE..U+27EF are additional, specifically-styled mathematical parentheses. Unlike the mathematical and CJK brackets just discussed, the flattened parentheses do not have corresponding wide CJK versions which they would need to be contrasted with.

Long Division. U+27CC LONG DIVISION is an operator intended for the representation of long division expressions, as may be seen in elementary and secondary school mathematical textbooks, for example. In use and rendering it shares some characteristics with U+221A SQUARE ROOT; in rendering, the top bar may be stretched to

extend over the top of the denominator of the division expression. Full support of such rendering may, however, require specialized mathematical software.

Fractional Slash and Other Diagonals. U+27CB MATHEMATICAL RISING DIAGONAL and U+27CD MATHEMATICAL FALLING DIAGONAL are limited-use mathematical symbols, to be distinguished from the more widely used solidi and reverse solidi operators encoded in the Basic Latin, Mathematical Operators, and Miscellaneous Mathematical Symbols-B blocks. Their glyphs are invariably drawn at a 45 degree angle, instead of the more upright slants typical for the solidi operators. The box drawing characters U+2571 and U+2572, whose glyphs may also be found at a 45 degree angle in some fonts, are not intended to be used as mathematical symbols. One usage recorded for U+27CB and U+27CD is in the notation for spaces of double cosets. The former corresponds to the LaTeX entity \diagup and the latter to \diagdown.

22.5.5 Miscellaneous Mathematical Symbols-B: U+2980–U+29FF

The Miscellaneous Mathematical Symbols-B block contains miscellaneous symbols used for mathematical notation, including fences and other delimiters. Some of the symbols in this block may also be used as operators in some contexts.

Wiggly Fence. U+29D8 LEFT WIGGLY FENCE has a superficial similarity to U+FE34 PRESENTATION FORM FOR VERTICAL WAVY LOW LINE. The latter is a wiggly sidebar character, intended for legacy support as a style of underlining character in a vertical text layout context; it has a compatibility mapping to U+005F LOW LINE. This represents a very different usage from the standard use of fence characters in mathematical notation.

Tiny and Miny. U+29FE TINY and U+29FF MINY are unary mathematical operators used in combinatorial game theory. TINY yields an infinitesimal positive value, while MINY yields an infinitesimal negative value. The glyphs for TINY and MINY resemble the plus sign and minus sign, respectively, but should be shown distinctly, with thickened ends to their bars.

22.5.6 Miscellaneous Symbols and Arrows: U+2B00–U+2BFF

The Miscellaneous Symbols and Arrows block contains more mathematical symbols and arrows. The arrows in this block extend and complete sets of arrows in other blocks. The other mathematical symbols complement various sets of geometric shapes. For a discussion of the use of such shape symbols in mathematical contexts, see "Geometric Shapes: U+25A0–U+25FF" and "Geometric Shapes Extended: U+1F780–U+1F7FF" in *Section 22.8, Geometrical Symbols*.

This block also contains various types of generic symbols. These complement the set of symbols in the Miscellaneous Symbols block, U+2600..U+26FF.

22.5.7 Arrows: U+2190–U+21FF

Arrows are used for a variety of purposes: to imply directional relation, to show logical derivation or implication, and to represent the cursor control keys.

Accordingly, the Unicode Standard includes a fairly extensive set of generic arrow shapes, especially those for which there are established usages with well-defined semantics. It does not attempt to encode every possible stylistic variant of arrows separately, especially where their use is mainly decorative. For most arrow variants, the Unicode Standard provides encodings in the two horizontal directions, often in the four cardinal directions. For the single and double arrows, the Unicode Standard provides encodings in eight directions.

Bidirectional Layout. In bidirectional layout, arrows are not automatically mirrored, because the direction of the arrow could be relative to the text direction or relative to an absolute direction. Therefore, if text is copied from a left-to-right to a right-to-left context, or vice versa, the character code for the desired arrow direction in the new context must be used. For example, it might be necessary to change U+21D2 RIGHTWARDS DOUBLE ARROW to U+21D0 LEFTWARDS DOUBLE ARROW to maintain the semantics of "implies" in a right-to-left context. For more information on bidirectional layout, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm."

Standards. The Unicode Standard encodes arrows from many different international and national standards as well as corporate collections.

Unifications. Arrows expressing mathematical relations have been encoded in the Arrows block as well as in the supplemental arrows blocks. An example is U+21D2 ⇒ RIGHTWARDS DOUBLE ARROW, which may be used to denote *implies*. Where available, such usage information is indicated in the annotations to individual characters in the code charts. However, because the arrows have such a wide variety of applications, there may be several semantic values for the same Unicode character value.

22.5.8 Supplemental Arrows

The Supplemental Arrows-A (U+27F0..U+27FF), Supplemental Arrows-B (U+2900.. U+297F), Miscellaneous Symbols and Arrows (U+2B00..U+2BFF), and Supplemental Arrows-C (U+1F800..U+1F8FF) blocks contain a large repertoire of arrows to supplement the main set in the Arrows block. Many of the supplemental arrows in the Miscellaneous Symbols and Arrows block, particularly in the range U+2B30..U+2B4C, are encoded to ensure the availability of left-right symmetric pairs of less common arrows, for use in bidirectional layout of mathematical text. The two arrows U+1F8C0 and U+1F8C1 are used in modern Egyptian hieroglyphic text materials to indicate the direction the glyphs are facing when the text is laid out in vertical

columns. Some of the arrows encoded in the Supplemental Arrows-C block are encoded to fill out sets of arrows for mapping to legacy computing character sets and the Smalltalk programming language.

Long Arrows. The long arrows encoded in the range U+27F5..U+27FF map to standard SGML entity sets supported by MathML. Long arrows represent distinct semantics from their short counterparts, rather than mere stylistic glyph differences. For example, the shorter forms of arrows are often used in connection with limits, whereas the longer ones are associated with mappings. The use of the long arrows is so common that they were assigned entity names in the ISOAMSA entity set, one of the suite of mathematical symbol entity sets covered by the Unicode Standard.

22.5.9 Standardized Variants of Mathematical Symbols

These mathematical variants are all produced with the addition of U+FE00 VARIATION SELECTOR-1 (VS1) to mathematical operator base characters. The valid combinations are listed in the file StandardizedVariants.txt in the Unicode Character Database. All combinations not listed there are unspecified and are reserved for future standardization; no conformant process may interpret them as standardized variants.

Change in Representative Glyphs for U+2278 and U+2279. In Version 3.2 of the Unicode Standard, the representative glyphs for U+2278 NEITHER LESS-THAN NOR GREATER-THAN and U+2279 NEITHER GREATER-THAN NOR LESS-THAN were changed from using a vertical cancellation to using a slanted cancellation. This change was made to match the long-standing canonical decompositions for these characters, which use U+0338 COMBINING LONG SOLIDUS OVERLAY. The symmetric forms using the vertical stroke continue to be acceptable glyph variants. Using U+2276 LESS-THAN OR GREATER-THAN or U+2277 GREATER-THAN OR LESS-THAN with U+20D2 COMBINING LONG VERTICAL LINE OVERLAY will display these variants explicitly. Unless fonts are created with the intention to add support for both forms, there is no need to revise the glyphs in existing fonts; the glyphic range implied by using the base character code alone encompasses both shapes. For more information, see *Section 23.4, Variation Selectors*.

22.6 Invisible Mathematical Operators

In mathematics, some operators and punctuation are often implied but not displayed. The General Punctuation block contains several special format control characters known as *invisible operators*, which can be used to make such operators explicit for use in machine interpretation of mathematical expressions. Use of invisible operators is optional and is intended for interchange with math-aware programs.

A more complete discussion of mathematical notation can be found in Unicode Technical Report #25, "Unicode Support for Mathematics."

Invisible Separator. U+2063 INVISIBLE SEPARATOR (also known as *invisible comma*) is intended for use in index expressions and other mathematical notation where two adjacent variables form a list and are not implicitly multiplied. In mathematical notation, commas are not always explicitly present, but they need to be indicated for symbolic calculation software to help it disambiguate a sequence from a multiplication. For example, the double $_{ij}$ subscript in the variable a_{ij} means $a_{i,\,j}$—that is, the i and j are separate indices and not a single variable with the name ij or even the product of i and j. To represent the implied list separation in the subscript $_{ij}$, one can insert a nondisplaying *invisible separator* between the i and the j. In addition, use of the invisible comma would hint to a math layout program that it should typeset a small space between the variables.

Invisible Multiplication. Similarly, an expression like mc^2 implies that the mass m multiplies the square of the speed c. To represent the implied multiplication in mc^2, one inserts a nondisplaying U+2062 INVISIBLE TIMES between the m and the c. Another example can be seen in the expression $f^{ij}(\cos(ab))$, which has the same meaning as $f^{ij}(\cos(a \times b))$, where $\times$ represents *multiplication*, not the *cross product*. Note that the spacing between characters may also depend on whether the adjacent variables are part of a list or are to be concatenated (that is, multiplied).

Invisible Plus. The invisible plus operator, U+2064 INVISIBLE PLUS, is used to unambiguously represent expressions like 3¼ which occur frequently in school and engineering texts. To ensure that 3¼ means 3 plus ¼—in uses where it is not possible to rely on a human reader to disambiguate the implied intent of juxtaposition—the invisible plus operator is used. In such uses, not having an operator at all would imply multiplication.

Invisible Function Application. U+2061 FUNCTION APPLICATION is used for an implied function dependence, as in $f(x + y)$. To indicate that this is the function f of the quantity $x + y$ and not the expression $fx + fy$, one can insert the nondisplaying *function application symbol* between the f and the left parenthesis.

22.7 Technical Symbols

22.7.1 Control Pictures: U+2400–U+243F

The need to show the presence of the C0 control codes unequivocally when data are displayed has led to conventional representations for these nongraphic characters.

Code Points for Pictures for Control Codes. By definition, control codes themselves are manifested only by their action. However, it is sometimes necessary to show the position of a control code within a data stream. Conventional illustrations for the ASCII C0 control codes have been developed—but the characters U+2400..U+241F and U+2424 are intended for use as unspecified graphics for the corresponding control codes. This choice allows a particular application to use *any* desired pictorial representation of the given control code. It assumes that the particular pictures used to represent control codes are often specific to different systems and are rarely the subject of text interchange between systems.

Pictures for ASCII Space. By definition, the SPACE is a blank graphic. Conventions have also been established for the visible representation of the space. Three specific characters are provided that may be used to visually represent the ASCII space character, U+2420 SYMBOL FOR SPACE, U+2422 BLANK SYMBOL, and U+2423 OPEN BOX.

Pictures for Delete. The control code for *delete* is sometimes depicted with a visible graphic representation. U+2421 SYMBOL FOR DELETE can be used to select a visible depiction for *delete*, with the exact graphical form left unspecified. In contrast, U+2425 SYMBOL FOR DELETE FORM TWO represents a specific form for a keyboard symbol from ISO 9995-7 associated with an *undoable delete* function. Several specific checkerboard or shade forms for *delete* in the range U+2427..U+2429, are encoded for compatibility mapping to sets of symbols for legacy computing, specifically for the Apple II, TRS-80, and Amstrad CPC legacy computer character sets.

Standards. The CNS 11643 standard encodes characters for pictures of control codes. Standard representations for control characters have been defined—for example, in ANSI X3.32 and ISO 2047. If desired, the characters U+2400..U+241F may be used for these representations.

## 22.7.2 Miscellaneous Technical: U+2300–U+23FF

This block encodes technical symbols, including keytop labels such as U+232B ERASE TO THE LEFT. Excluded from consideration were symbols that are not normally used in one-dimensional text but are intended for two-dimensional diagrammatic use, such as most symbols for electronic circuits.

Keytop Labels. Where possible, keytop labels have been unified with other symbols of like appearance—for example, U+21E7 UPWARDS WHITE ARROW to indicate the Shift key. While symbols such as U+2318 PLACE OF INTEREST SIGN and U+2388 HELM SYMBOL are generic symbols that have been adapted to use on keytops, other symbols specifically follow ISO/IEC 9995-7.

Floor and Ceiling. The floor and ceiling symbols encoded at U+2308..U+230B are tall, narrow mathematical delimiters. These symbols should not be confused with the CJK corner brackets at U+300C and U+300D, which are wide characters used as quotation marks in East Asian text. They should also be distinguished from the half brackets at U+2E22..U+2E25, which are the most generally used editorial marks shaped like corner brackets. Additional types of editorial marks, including further corner bracket forms, can be found in the Supplemental Punctuation block (U+2E00..U+2E7F).

Crops and Quine Corners. Crops and quine corners are most properly used in two-dimensional layout but may be referred to in plain text. This usage is shown in *Figure 22-9*.

Figure 22-9. Usage of Crops and Quine Corners

Crops   *Quine corners*
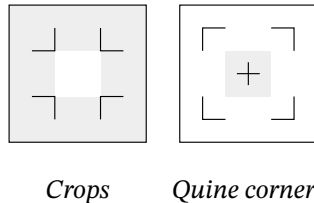
Angle Brackets. U+2329 LEFT-POINTING ANGLE BRACKET and U+232A RIGHT-POINTING ANGLE BRACKET have long been canonically equivalent to the CJK punctuation characters U+3008 LEFT ANGLE BRACKET and U+3009 RIGHT ANGLE BRACKET, respectively. This canonical equivalence implies that the use of the latter (CJK) code points is preferred and that U+2329 and U+232A are also "wide" characters. (See Unicode Standard Annex #11, "East Asian Width," for the definition of the East Asian wide property.) For this reason, the use of U+2329 and U+232A is deprecated for mathematics and for technical publication, where the wide property of the characters has the potential to interfere with the proper formatting of mathematical formulae. The angle brackets specifically provided for mathematics, U+27E8 MATHE-MATICAL LEFT ANGLE BRACKET and U+27E9 MATHEMATICAL RIGHT ANGLE BRACKET, should be used instead. See *Section 22.5, Mathematical Symbols*.

APL Functional Symbols. APL (A Programming Language) makes extensive use of functional symbols constructed by composition with other, more primitive functional symbols. It used backspace and overstrike mechanisms in early computer implementations. In principle, functional composition is productive in APL; in practice, a rela-

tively small number of composed functional symbols have become standard operators in APL. This relatively small set is encoded in its entirety in this block. All other APL extensions can be encoded by composition of other Unicode characters. For example, the APL symbol *a underbar* can be represented by U+0061 LATIN SMALL LETTER A + U+0332 COMBINING LOW LINE.

Symbol Pieces. The characters in the range U+239B..U+23B3, plus U+23B7, constitute a set of bracket and other symbol fragments for use in mathematical typesetting. These pieces originated in older font standards but have been used in past mathematical processing as characters in their own right to make up extra-tall glyphs for enclosing multiline mathematical formulae. Mathematical fences are ordinarily sized to the content that they enclose. However, in creating a large fence, the glyph is not scaled proportionally; in particular, the displayed stem weights must remain compatible with the accompanying smaller characters. Thus simple scaling of font outlines cannot be used to create tall brackets. Instead, a common technique is to build up the symbol from pieces. In particular, the characters U+239B LEFT PARENTHESIS UPPER HOOK through U+23B3 SUMMATION BOTTOM represent a set of glyph pieces for building up large versions of the fences ( ), [ ], and { }, and of the large operators $\sum$ and $\int$. These brace and operator pieces are compatibility characters. They should not be used in stored mathematical text, although they are often used in the data stream created by display and print drivers.

Table 22-6 shows which pieces are intended to be used together to create specific symbols. For example, an instance of U+239B can be positioned relative to instances of U+239C and U+239D to form an extra-tall (three or more line) left parenthesis. The center sections encoded here are meant to be used only with the top and bottom pieces encoded adjacent to them because the segments are usually graphically constructed within the fonts so that they match perfectly when positioned at the same *x* coordinates.

Table 22-6. Use of Mathematical Symbol Pieces

	Two-Row	Three-Row	Five-Row
Summation	23B2, 23B3		
Integral	2320, 2321	2320, 23AE, 2321	2320, 3×23AE, 2321
Left parenthesis	239B, 239D	239B, 239C, 239D	239B, 3×239C, 239D
Right parenthesis	239E, 23A0	239E, 239F, 23A0	239E, 3×239F, 23A0
Left bracket	23A1, 23A3	23A1, 23A2, 23A3	23A1, 3×23A2, 23A3
Right bracket	23A4, 23A6	23A4, 23A5, 23A6	23A4, 3×23A5, 23A6
Left brace	23B0, 23B1	23A7, 23A8, 23A9	23A7, 23AA, 23A8, 23AA, 23A9
Right brace	23B1, 23B0	23AB, 23AC, 23AD	23AB, 23AA, 23AC, 23AA, 23AD

Horizontal Brackets. In mathematical equations, delimiters are often used horizontally, where they expand to the width of the expression they encompass. The six

bracket characters in the range U+23DC..U+23E1 can be used for this purpose. In the context of mathematical layout, U+23B4 TOP SQUARE BRACKET and U+23B5 BOTTOM SQUARE BRACKET are also used that way. For more information, see Unicode Technical Report #25, "Unicode Support for Mathematics."

The set of horizontal square brackets, U+23B4 TOP SQUARE BRACKET and U+23B5 BOTTOM SQUARE BRACKET, together with U+23B6 BOTTOM SQUARE BRACKET OVER TOP SQUARE BRACKET, are used by certain legacy applications to delimit vertical runs of text in non-CJK terminal emulation. U+23B6 is used where a single character cell is both the end of one such run and the start of another. The use of these characters in terminal emulation should not be confused with the use of rotated forms of brackets for vertically rendered CJK text. See the further discussion of this issue in *Section 6.2, General Punctuation*.

Decimal Exponent Symbol. U+23E8 DECIMAL EXPONENT SYMBOL is for compatibility with the Russian standard GOST 10859-64, as well as the paper tape and punch card standard, Alcor (DIN 66006). It represents a fixed token introducing the exponent of a real number in scientific notation, comparable to the more common usage of "e" in similar notations: 1.621e5. It was used in the early computer language ALGOL-60, and appeared in some Soviet-manufactured computers, such as the BESM-6 and its emulators. In the Unicode Standard it is treated simply as an atomic symbol; it is not considered to be equivalent to a generic subscripted form of the numeral "10" and is not given a decomposition. The vertical alignment of this symbol is slightly lower than the baseline, as shown in *Figure 22-10*.

Figure 22-10. Usage of the Decimal Exponent Symbol

```
СИСТЕМА АЛГОЛ-БЭСМ6. ВАРИАНТ 01-05-79.
СЧЕТ БЕЗ КОНТРОЛЯ
1. _BEGIN OUTPUT('E', 355.0/113.0) _END
------------------------------------------------------------
.3141592920₁₀+01
```

Dental Symbols. The set of symbols from U+23BE to U+23CC form a set of symbols from JIS X 0213 for use in dental notation.

Metrical Symbols. The symbols in the range U+23D1..U+23D9 are a set of spacing symbols used in the metrical analysis of poetry and lyrics.

Electrotechnical Symbols. The Miscellaneous Technical block also contains a smattering of electrotechnical symbols. These characters are not intended to constitute a complete encoding of all symbols used in electrical diagrams, but rather are compatibility characters encoded primarily for mapping to other standards. The symbols in the range U+238D..U+2394 are from the character set with the International Registry number 181. U+23DA EARTH GROUND and U+23DB FUSE are from HKSCS-2001.

User Interface Symbols. The characters U+231A, U+231B, and U+23E9 through U+23FA are often found in user interfaces for media players, clocks, alarms, and timers, as well as in text discussing those user interfaces. The black medium triangles (U+23F4..U+23F7) are the preferred shapes for User Interface purposes, rather than the similar geometric shapes located in the Geometric Shapes block: U+25A0..U+25FF. The Miscellaneous Symbols and Pictographs block also contains many user interface symbols in the ranges U+1F500..U+1F518, U+1F53A..U+1F53D and U+1F5BF..U+1F5DD, as well as clock face symbols in the range U+1F550..U+1F567.

Standards. This block contains a large number of symbols from ISO/IEC 9995-7:1994, *Information technology—Keyboard layouts for text and office systems—Part 7: Symbols used to represent functions.*

ISO/IEC 9995-7 contains many symbols that have been unified with existing and closely related symbols in Unicode. These symbols are shown with their ordinary shapes in the code charts, not with the particular glyph variation required by conformance to ISO/IEC 9995-7. Implementations wishing to be conformant to ISO/IEC 9995-7 in the depiction of these symbols should make use of a suitable font.

22.7.3 Optical Character Recognition: U+2440–U+245F

This block includes those symbolic characters of the OCR-A character set that do not correspond to ASCII characters, as well as magnetic ink character recognition (MICR) symbols used in check processing.

Standards. Both sets of symbols are specified in ISO 2033.

22.7.4 Symbols for Legacy Computing: U+1FB00-U+1FBFF

Symbols for Legacy Computing Supplement: U+1CC00–U+1CEBF

The Unicode Standard encodes an extensive set of symbols to support legacy computing graphic characters, primarily located in these blocks.

Support for these legacy computing symbols includes 212 characters added to the Symbols for Legacy Computing block in Version 13.0 to provide compatibility with a wide range of early home computers, or "microcomputers," manufactured from the mid-1970s to the mid-1980s. These symbols also cover the teletext broadcasting standard originally developed in the early 1970s, and the Minitel standard developed in the 1980s. This collection of early microcomputer symbols includes support for the character sets of Amstrad CPC, Apple 8-bit, Atari 8 and 16-bit, Commodore 8 and 16-bit, MSX, Yamaha, RISC OS, and Tandy. Version 16.0 added 37 characters (U+1FBCB..U+1FBEF) that were employed on lesser-known legacy computer platforms, including Mattel Aquarius and the Sharp MX series.

The Symbols for Legacy Computing Supplement block contains 680 symbols from the lesser-known platforms and four characters from the Smalltalk programming language (U+1CEB0..U+1CEB3).

Most of the symbols in these blocks are semi-graphics: block-style symbols which can be combined to simulate an all-points-addressable graphic display. Many platforms used these semi-graphic characters to support a graphics mode: small blocks that would be plotted at various coordinates, resulting in the appropriate full-sized block character consisting of the necessary "on" and "off" blocks. Other symbols in these blocks include box drawing and shading characters, miscellaneous arrows, game sprites, and stick figures. In the teletext specification, symbols in this group can be displayed either with cells joined together or with a narrow space between cells.

A set of ASCII digits 0 through 9 (U+1FBF0..U+1FBF9) are styled as upright seven-segment digits to match the ones that were often used in Atari 16-bit applications for game scores. Outlined uppercase Latin letters and ASCII digits from the European character set for the Sharp MZ-series machines are encoded in the range U+1CCD6..U+1CCF9.

Terminal graphics legacy symbols are also encoded in the Miscellaneous Technical block. They include block-style semi-graphics, border-colored characters, and box drawing characters. Other box drawing symbols are encoded in the Box Drawing block and several characters are unified with characters in the Block Elements block.

In particular, the Miscellaneous Technical block includes the horizontal scan line characters, U+23BA HORIZONTAL SCAN LINE-1 through U+23BD HORIZONTAL SCAN LINE-9, which represent characters that were encoded in character ROM for use with nine-line character graphic cells. Horizontal scan line characters are encoded for scan lines 1, 3, 7, and 9. The horizontal scan line character for scan line 5 is unified with U+2500 BOX DRAWINGS LIGHT HORIZONTAL.

The symbols in these blocks combined with the vertical and horizontal line characters encoded in the Miscellaneous Technical block provide the compatibility characters needed for applications to emulate various early computer terminal support.

22.8 Geometrical Symbols

Geometrical symbols are a collection of geometric shapes and their derivatives plus block elements and characters used for box drawing in legacy environments. In addition to the blocks described in this section, the Miscellaneous Technical (U+2300..U+23FF), Miscellaneous Symbols (U+2600..U+26FF), and Miscellaneous Symbols and Arrows (U+2B00.. U+2BFF) blocks contain geometrical symbols that complete the set of shapes in the Geometric Shapes block.

22.8.1 Box Drawing and Block Elements

Box drawing and block element characters are graphic compatibility characters in the Unicode Standard. A number of existing national and vendor standards, including IBM PC Code Page 437, contain sets of characters intended to enable a simple kind of display cell graphics, assuming terminal-type screen displays of fixed-pitch character cells. The Unicode Standard does not encourage this kind of character-cell-based graphics model, but does include sets of such characters for backward compatibility with the existing standards.

Box Drawing. The Box Drawing block (U+2500..U+257F) contains a collection of graphic compatibility characters that originate in legacy standards in use prior to 1990 and that are intended for drawing boxes of various shapes and line widths for user interface components in character-cell-based graphic systems.

The "light," "heavy," and "double" attributes for some of these characters reflect the fact that the original sets often had a two-way distinction, between a light versus heavy line or a single versus double line, and included sufficient pieces to enable construction of graphic boxes with distinct styles that abutted each other in display.

In particular, the mappings to some Videotex mosaic drawing characters noted in the code charts refer to the concept of "heavy" as specified in early Videotex character registrations and Recommendations, which made a two-way distinction between light and heavy. See, for example, ITU-T Recommendation T.101, International Inter-working for Videotex Services (November, 1988). The mappings do not reflect later Videotex registrations and modifications to the relevant Recommendations which specified three levels of weight distinction in lines for box drawing characters.

The lines in the box drawing characters typically extend to the middle of the top, bottom, left, and/or right of the bounding box for the character cell. They are designed to connect together into continuous lines, with no gaps between them. When emulating terminal applications, fonts that implement the box drawing characters should do likewise.

Block Elements. The Block Elements block (U+2580..U+259F) contains another collection of graphic compatibility characters. Unlike the box drawing characters, the legacy block elements are designed to fill some defined fraction of each display cell or to fill each display cell with some defined degree of shading. These elements were used to create crude graphic displays in terminals or in terminal modes on displays where bit-mapped graphics were unavailable.

Half-block fill characters are included for each half of a display cell, plus a graduated series of vertical and horizontal fractional fills based on one-eighth parts. The fractional fills do not form a logically complete set but are intended only for backward compatibility. There is also a set of quadrant fill characters, U+2596..U+259F, which are designed to complement the half-block fill characters and U+2588 FULL BLOCK. When emulating terminal applications, fonts that implement the block element characters should be designed so that adjacent glyphs for characters such as U+2588 FULL BLOCK create solid patterns with no gaps between them.

Standards. The box drawing and block element characters were derived from GB 2312, KS X 1001, a variety of industry standards, and several terminal graphics sets. The Videotex Mosaic characters, which have similar appearances and functions, are unified against these sets.

22.8.2 Geometric Shapes: U+25A0–U+25FF

The Geometric Shapes are a collection of characters intended to encode prototypes for various commonly used geometrical shapes—mostly squares, triangles, and circles. The collection is somewhat arbitrary in scope; it is a compendium of shapes from various character and glyph standards. The typical distinctions more systematically encoded include black versus white, large versus small, basic shape (square versus triangle versus circle), orientation, and top versus bottom or left versus right part.

Hatched Squares. The hatched and cross-hatched squares at U+25A4..U+25A9 are derived from the Korean national standard (KS X 1001), in which they were probably intended as representations of fill patterns. Because the semantics of those characters are insufficiently defined in that standard, the Unicode character encoding simply carries the glyphs themselves as geometric shapes to provide a mapping for the Korean standard.

Lozenge. U+25CA ◊ LOZENGE is a typographical symbol seen in PostScript and in the Macintosh character set. It should be distinguished from both the generic U+25C7 WHITE DIAMOND and the U+2662 WHITE DIAMOND SUIT, as well as from another character sometimes called a lozenge, U+2311 SQUARE LOZENGE.

Use in Mathematics. Many geometric shapes are used in mathematics. When used for this purpose, the center points of the glyphs representing geometrical shapes

should line up at the center line of the mathematical font. This differs from the alignment used for some of the representative glyphs in the code charts.

For several simple geometrical shapes—circle, square, triangle, diamond, and lozenge—differences in size carry semantic distinctions in mathematical notation, such as the difference between use of the symbol as a variable or as one of a variety of operator types. The Miscellaneous Symbols and Arrows block contains numerous characters representing other sizes of these geometrical symbols. Several other blocks, such as General Punctuation, Mathematical Operators, Block Elements, Miscellaneous Symbols, and Geometric Shapes Extended, contain a few other characters which are members of the size-graded sets of such symbols.

For more details on the use of geometrical shapes in mathematics, see Unicode Technical Report #25, "Unicode Support for Mathematics."

Standards. The Geometric Shapes are derived from a large range of national and vendor character standards. The squares and triangles at U+25E7..U+25EE are derived from the Linotype font collection. U+25EF LARGE CIRCLE is included for compatibility with the JIS X 0208-1990 Japanese standard.

22.8.3 Geometric Shapes Extended: U+1F780–U+1F7FF

The repertoire for the Geometric Shapes Extended block mostly originates from the set of Wingdings. It is intended primarily to complement existing sets of geometric shape symbols in other blocks. The choice of sizes for this extension is done with the goal that font designers will be able to scale uniformly among the various sizes for each class of geometric shapes. *Table 22-7* provides a list of the sets that have characters spanning multiple blocks, including the Geometric Shapes Extended block. Differences in size may carry semantic distinction in mathematical notation.

Table 22-7. Geometric Shape Collections

Description	Code Points
Black circles	22C5, 2219, 1F784, 2022, 2981, 26AB, 25CF, 2B24
White circles	25CB, 2B58, 1F785..1F789
Colored circles	26AA, 26AB, 1F534, 1F535, 1F7E0..1F7E4
Black squares	1F78C, 2B1D, 1F78D, 25AA, 25FE, 25FC, 25A0, 2B1B
White squares	25A1, 1F78E..1F792
Colored squares	2B1C, 2B1B, 1F7E5..1F7EB
White squares containing another black square	1F794, 25A3, 1F795
Black diamonds	1F797, 1F798, 2B29, 1F799, 2B25, 25C6

White diamonds containing another black diamond	1F79A, 25C8, 1F79B
Black lozenges	1F79D, 1F79E, 2B2A, 1F79F, 2B27, 29EB
Five pointed stars	1F7C9, 2605, 1F7CA, 272F
Six pointed stars	2736, 1F7CB..1F7CD
Eight pointed stars	2735, 1F7CE..1F7D1
Twelve pointed stars	1F7D2, 2739, 1F7D3, 1F7D4

This block also contains a set of colored circles and squares in the range U+1F7E0..U+1F7EB. Those colored circles and squares are intended for use with emoji, to augment the colored circles and other colored sets for emoji. *Table 22-7* shows these sets, including white and black circles and squares, and red and blue circles from other blocks. Those sets are listed in the order: white, black, red, blue, orange, yellow, green, purple, brown. Unlike emoji modifiers for skin tone (see Unicode Technical Standard #51, "Unicode Emoji"), the symbols for colored circles and squares are simply graphical symbols which may convey the concepts of colors, but with no immediate implications for rendering of glyphs with those particular colors. For example, a user could specify a yellow circle symbol together with a ribbon emoji symbol to convey the notion of a "yellow ribbon," but there would be no expectation that the font would combine the two characters and draw an actual yellow ribbon. These colored circles and squares are often used decoratively in emoji text, with no other semantic intent.

22.9 Miscellaneous Symbols

There are numerous blocks defined in the Unicode Standard which contain miscellaneous symbols that do not fit well into any of the categories of symbols already discussed. These include various small sets of special-use symbols such as zodiacal symbols, map symbols, symbols used in transportation and accommodation guides, dictionary symbols, gender symbols, and so forth. There are additional larger sets, such as sets of symbols for game pieces or playing cards, and divination symbols associated with the Yijing or other texts, as well as sets of medieval or ancient symbols used only in historical contexts.

Of particular note are the large number of pictographic symbols, called *emoji* ("picture character"), in common use on mobile devices. Many emoji originated from character sets, called carrier sets, in early widespread use in cell phones in Japan. A number of other symbols are commonly shown with an emoji-like presentation. The majority of such symbols are encoded in the blocks listed in <u>Table 22-8</u>, but many emoji characters are encoded in other blocks. For a complete listing of the historic carrier emoji sets, including information about which of those emoji characters have been unified with other symbol characters in the Unicode Standard, see the data file EmojiSources.txt in the Unicode Character Database. The list of all Unicode characters that normally may be candidates for emoji presentation can be found in the data file emoji-data.txt in the Unicode Character Database.

Table 22-8. Blocks with Characters Often Shown as Emoji

Range	Block Name
2600..26FF	Miscellaneous Symbols
1F300..1F5FF	Miscellaneous Symbols and Pictographs
1F600..1F64F	Emoticons
1F680..1F6FF	Transport and Map Symbols
1F900..1F9FF	Supplemental Symbols and Pictographs
1FA70..1FAFF	Symbols and Pictographs Extended-A

An additional category of miscellaneous symbols are the so-called *dingbat* characters. These are essentially compatibility characters representing very specific glyph shapes associated with common "symbol" fonts in widespread legacy use. Symbols identified as "dingbats" are encoded in various blocks. The well-known "Zapf Dingbats" set is encoded comprehensively in the Dingbats block, U+2700..U+27BF. Other sets of dingbats, such as the Wingdings and Webdings sets, are encoded in various symbol blocks, but the majority are found in the Miscellaneous Symbols and Pictographs block, U+1F300..U+1F5FF.

Corporate logos and collections of graphical elements or pictures are not included in the Unicode Standard, because they tend either to be very specific in usage (logos, political party symbols, and so on) or are nonconventional in appearance and semantic interpretation (clip art collections), and hence are inappropriate for encoding as characters. The Unicode Standard recommends that such items be incorporated in text via higher-level protocols that allow intermixing of graphic images with text, rather than by indefinite extension of the number of miscellaneous symbols encoded as characters. Newer emoji-like symbols using embedded graphics are already in widespread use on mobile phones and other devices.

Rendering of Emoji. Many of the characters in the blocks listed in *Table 22-8* are often presented in an emoji style. There may be a great deal of variability in presentation, along three axes:

- Glyph shape: Emoji may have a great deal of flexibility in the choice of glyph shape used to render them.

- Color: Many characters in an emoji context (such as cell phone e-mail or text messages) are displayed in color, sometimes as a multicolor image. While this is particularly true of emoji, there are other cases where non-emoji symbols, such as game symbols, may be displayed in color.

- Animation: Some characters in an emoji context are presented in animated form, usually as a repeating sequence of two to four images.

Emoji may be presented using color or animation, but need not be. Because many characters in the carrier emoji sets or other sources are unified with Unicode characters that originally came from other sources, it may not always be clear whether a character should be presented using an emoji style. However, for most such characters, variation sequences have been defined which can specify text or emoji presentation. Unicode Technical Standard #51, "Unicode Emoji," provides some guidance about which characters should have which presentation style in various environments.

Color Words in Unicode Character Names. The representative glyph shown in the code charts for a character is always monochrome. The character name may include a term such as BLACK or WHITE, or in the case of characters for emoji pictographs, other color terms such as BLUE or ORANGE. The use of BLACK or WHITE in names such as BLACK MEDIUM SQUARE or WHITE MEDIUM SQUARE is generally intended to contrast filled versus outline shapes, or a darker color fill versus a lighter color fill; it is not intended to suggest that the character must be presented in black or white, respectively. Similarly, the color terms in names such as BLUE HEART or ORANGE BOOK are intended to help identify the characters; the characters may be presented using color, or in monochrome using different styles of shading or crosshatching, for example.

In Version 12.0 of the Unicode Standard, seven large, colored square emoji were added in the range U+1F7E5..U+1F7EB. Along with the earlier encoded U+2B1B BLACK LARGE SQUARE and U+2B1C WHITE LARGE SQUARE, these colored square emoji may be used in emoji ZWJ sequences to indicate that a base emoji should be displayed with the color of the square, if possible. The color of the square emoji is a general hint, and the color of the resulting image for the emoji ZWJ sequence need not be exactly the same as the colored square displayed by itself. Only a small number of such sequences are in the set of emoji sequences recommended for general interchange (RGI). See emoji-zwj-sequences.txt, documented in Annex A of Unicode Technical Standard #51, "Unicode Emoji."

22.9.1 Miscellaneous Symbols and Pictographs

The Miscellaneous Symbols (U+2600..U+26FF), Miscellaneous Symbols and Pictographs (U+1F300..U+1F5FF), Supplemental Symbols and Pictographs (U+1F900..U+1F9FF), and Symbols and Pictographs Extended-A (U+1FA70..U+1FAFF) blocks contain very heterogeneous collections of symbols that do not fit in any other Unicode character block and that tend to be pictographic in nature. These symbols are typically used for text decorations, but they may also be treated as normal text characters in applications such as typesetting chess books, card game manuals, and horoscopes.

The order of symbols in these blocks is arbitrary, but an attempt has been made to keep like symbols together and to group subsets of them into meaningful orders. Some of these subsets include weather and astronomical symbols, pointing hands, religious and ideological symbols, the Yijing (I Ching) trigrams, planet and zodiacal symbols, game symbols, musical dingbats, and recycling symbols. (For other moon phases, see the circle-based shapes in the Geometric Shapes block.)

Standards. The symbols in these blocks are derived from a large range of national and vendor character standards. Among them, characters from the Japanese Association of Radio Industries and Business (ARIB) standard STD-B24 are widely represented in the Miscellaneous Symbols block. The symbols from ARIB were initially used in the context of digital broadcasting, but in many cases their usage has evolved to more generic purposes. The Miscellaneous Symbols and Pictographs block includes many characters from the carrier emoji sets and the Wingdings/Webdings collections.

Weather Symbols. The characters in the ranges U+2600..U+2603, U+26C4..U+26CB, and U+1F321..U+1F32C, as well as U+2614 UMBRELLA WITH RAIN DROPS are weather symbols. These commonly occur as map symbols or in other contexts related to weather forecasting in digital broadcasting or on websites.

Moon and Sun Symbols. There are a variety of moon and sun symbols encoded in the Miscellaneous Symbols block (U+2609, U+263C..U+263E) and in the Miscella-
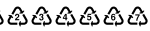
neous Symbols and Pictographs block (U+1F311..U+1F31E). Some of these are used in astrological charts, while others are merely playful symbols showing faces. Various crescent signs for the moon do not necessarily represent particular phases of the moon.

The moon symbols in the range U+1F311..U+1F318, in particular, represent a systematic set of eight symbols for the phases of the moon. These symbols appear, for example, in moon charts, almanacs, tide tables, and similar documents to represent particular phases of the moon. There is a notable difference in interpretation of symbols for phases of the moon between Northern Hemisphere users and Southern Hemisphere users, with the graphical orientation of waxing and waning phases reversed. So, for example, in the Southern Hemisphere, U+1F312 WAXING CRESCENT MOON SYMBOL would usually be interpreted as representing the *waning* crescent moon, instead.

The use of these moon symbols (U+1F311..U+1F318) should follow the *shape* of the graphic symbols, as shown in the code charts. Users should not simply assume from the character names that the symbols are intended to represent astronomical positions of the moon.

Traffic Signs. In general, traffic signs are quite diverse, tend to be elaborate in form and differ significantly between countries and locales. For the most part they are inappropriate for encoding as characters. However, there are a small number of conventional symbols which have been used as characters in contexts such as digital broadcasting or mobile phones. The characters in the ranges U+26CC..U+26CD and U+26CF..U+26E1 are traffic sign symbols of this sort, encoded for use in digital broadcasting. Additional traffic signs are in included in the Transport and Map Symbols block.

Dictionary and Map Symbols. The characters in the range U+26E8..U+26FF are dictionary and map symbols used in the context of digital broadcasting. Numerous other symbols in this block and scattered in other blocks also have conventional uses as dictionary or map symbols. For example, these may indicate special uses for words, or indicate types of buildings, points of interest, particular activities or sports, and so on.

Plastic Bottle Material Code System. The seven numbered logos encoded from U+2673 to U+2679, ♳♴♵♶♷♸♹, are from "The Plastic Bottle Material Code System," which was introduced in 1988 by the Society of the Plastics Industry (SPI). This set consistently uses thin, two-dimensional curved arrows suitable for use in plastics molding. In actual use, the symbols often are combined with an abbreviation of the material class below the triangle. Such abbreviations are not universal; therefore, they are not present in the representative glyphs in the code charts.

Recycling Symbol for Generic Materials. An unnumbered plastic resin code symbol U+267A ♲ RECYCLING SYMBOL FOR GENERIC MATERIALS is not formally part of the SPI system but is found in many fonts. Occasional use of this symbol as a generic materials code symbol can be found in the field, usually with a text legend below, but sometimes also surrounding or overlaid by other text or symbols. Sometimes the UNIVERSAL RECYCLING SYMBOL is substituted for the generic symbol in this context.

Universal Recycling Symbol. The Unicode Standard encodes two common glyph variants of this symbol: U+2672 ♲ UNIVERSAL RECYCLING SYMBOL and U+267B ♻ BLACK UNIVERSAL RECYCLING SYMBOL. Both are used to indicate that the material is recyclable. The white form is the traditional version of the symbol, but the black form is sometimes substituted, presumably because the thin outlines of the white form do not always reproduce well.

Paper Recycling Symbols. The two paper recycling symbols, U+267C ♼ RECYCLED PAPER SYMBOL and U+267D ♽ PARTIALLY-RECYCLED PAPER SYMBOL, can be used to distinguish between fully and partially recycled fiber content in paper products or packaging. They are usually accompanied by additional text.

Gender Symbols. The characters in the range U+26A2..U+26A9 are gender symbols. These are part of a set with U+2640 FEMALE SIGN, U+2642 MALE SIGN, U+26AA MEDIUM WHITE CIRCLE, and U+26B2 NEUTER. They are used in sexual studies and biology, for example. Some of these symbols have other uses as well, as astrological or alchemical symbols.

Genealogical Symbols. The characters in the range U+26AD..U+26B1 are sometimes seen in genealogical tables, where they indicate marriage and burial status. They may be augmented by other symbols, including the small circle indicating betrothal.

Game Symbols. The Miscellaneous Symbols block also contains a variety of small symbol sets intended for the representation of common game symbols or tokens in text. These include symbols for playing card suits, often seen in manuals for bridge and other card games, as well as a set of dice symbols. The chess symbols are often seen in figurine algebraic notation. In addition, there are symbols for game pieces or notation markers for go, shogi (Japanese chess), and draughts (checkers).

Larger sets of game symbols are encoded in their own blocks. See the discussion of playing cards, chess symbols, mahjong tile symbols, and domino tile symbols later in this section.

Animal Symbols. The animal symbol characters in the range U+1F400..U+1F42C are encoded primarily to cover the emoji sets used by Japanese cell phone carriers. Animal symbols are widely used in Asia as signs of the zodiac, and that is part of the reason for their inclusion in the cell phone sets. However, the particular animal

symbols seen in Japan and China are not the only animals used as zodiacal symbols throughout Asia. The set of animal symbols encoded in this block includes other animal symbols used as zodiacal symbols in Vietnam, Thailand, Persia, and other Asian countries. These zodiacal uses are specifically annotated in the Unicode code charts.

Other animal symbols have no zodiacal associations, and are included simply to cover the carrier emoji sets. A few of the animal symbols have conventional uses to designate types of meat on menus. Later additions of animal symbols fill perceived gaps in the set, responding to the wide popularity of animal symbols in Unicode-based emoji implementations.

Cultural Symbols. The five cultural symbols encoded in the range U+1F5FB..U+1F5FF mostly designate cultural landmarks of particular importance to Japan. They are encoded for compatibility with emoji sets used by Japanese cell phone carriers, and are not intended to set a precedent for encoding additional sets of cultural landmarks or other pictographic cultural symbols as characters.

Hand Symbols. The pictographic symbols for hands encoded in the ranges U+1F90F, U+1F918..U+1F91F, U+1F446..U+1F450, and U+1F58E..U+1F5A3, as well as in the U+270A..U+270D range in the Dingbats block, represent various hand gestures. The interpretations associated with such gestures vary significantly among cultures.

Emoji Modifiers. The emoji modifiers U+1F3FB..U+1F3FF designate five different skin tones based on the Fitzpatrick scale. These may be displayed in isolation as color or halftone swatches, or they may form a ligature with a preceding emoji character representing a person or body part in order to specify a particular appearance for that character.

Miscellaneous Symbols in Other Blocks. In addition to the blocks described in this section, which are devoted entirely to sets of miscellaneous symbols, there are many other blocks which contain small numbers of otherwise uncategorized symbols. See, for example, the Miscellaneous Symbols and Arrows block U+2B00..U+2B7F, the Enclosed Alphanumeric Supplement block U+1F100..U+1F1FF, the CJK Symbols and Punctuation block U+3000..U+303F, and the Ideographic Symbols and Punctuation block 16FE0..16FFF. Some of these blocks contain symbols which extend or complement sets of symbols contained in the Miscellaneous Symbols block.

22.9.2 Emoticons: U+1F600–U+1F64F

Emoticons (from "emotion" plus "icon") originated as a way to convey emotion or attitude in e-mail messages, using ASCII character combinations such as :-) to indicate a smile—and by extension, a joke—and :-(to indicate a frown. In East Asia, a

number of more elaborate sequences have been developed, such as (")(-_-)(") showing an upset face with hands raised.

Over time, many systems began replacing such sequences with images, and also began providing a way to input emoticon images directly, such as a menu or palette. The carrier emoji sets used by Japanese cell phone providers contain a large number of characters for emoticon images, and most of the characters in this block are from those sets. They are divided into a set of humanlike faces, a smaller set of cat faces that parallel some of the humanlike faces, and a set of gesture symbols that combine a human or monkey face with arm and hand positions.

Several emoticons are also encoded in the Miscellaneous Symbols block at U+2639.. U+263B and in the Supplemental Symbols and Pictographs block at U+1F910..U+1F917 and U+1F920..1F927.

22.9.3 Transport and Map Symbols: U+1F680–U+1F6FF

This block is similar to the blocks Miscellaneous Symbols and Miscellaneous Symbols and Pictographs, but is a more cohesive set of symbols. Many of these symbols originated in the emoji sets used by Japanese cell phone carriers.

Various traffic signs and map symbols are also encoded in the Miscellaneous Symbols block.

22.9.4 Dingbats: U+2700–U+27BF

Most of the characters in the Dingbats block are derived from a well-established set of glyphs, the ITC Zapf Dingbats series 100, which constitutes the industry standard "Zapf Dingbat" font currently available in most laser printers. The order of the Dingbats block basically follows the PostScript encoding. Dingbat characters derived from the Wingdings and Webdings sets are encoded in other blocks, particularly in the Miscellaneous Symbols and Pictographs block, U+1F300..U+1F5FF.

Unifications and Additions. Where a dingbat from the ITC Zapf Dingbats series 100 could be unified with a generic symbol widely used in other contexts, only the generic symbol was encoded. Examples of such unifications include card suits, BLACK STAR, BLACK TELEPHONE, and BLACK RIGHT POINTING INDEX (see the Miscellaneous Symbols block); BLACK CIRCLE and BLACK SQUARE (see the Geometric Shapes block); white encircled numbers 1 to 10 (see the Enclosed Alphanumerics block); and several generic arrows (see the Arrows block). Those four entries appear elsewhere in this chapter. Other dingbat-like characters, primarily from the carrier emoji sets, are encoded in the gaps that resulted from this unification.

In other instances, other glyphs from the ITC Zapf Dingbats series 100 glyphs have come to be recognized as having applicability as generic symbols, despite having originally been encoded in the Dingbats block. For example, the series of negative (black) circled numbers 1 to 10 are now treated as generic symbols for this sequence, the continuation of which can be found in the Enclosed Alphanumerics block. Other examples include U+2708 AIRPLANE and U+2709 ENVELOPE, which have definite semantics independent of the specific glyph shape, and which therefore should be considered generic symbols rather than symbols representing only the Zapf Dingbats glyph shapes.

For many of the remaining characters in the Dingbats block, their semantic value is primarily their shape; unlike characters that represent letters from a script, there is no well-established range of typeface variations for a dingbat that will retain its identity and therefore its semantics. It would be incorrect to arbitrarily replace U+279D TRIANGLE-HEADED RIGHTWARDS ARROW with any other right arrow dingbat or with any of the generic arrows from the Arrows block (U+2190..U+21FF). However, exact shape retention for the glyphs is not always required to maintain the relevant distinctions. For example, ornamental characters such as U+2741 EIGHT PETALLED OUTLINED BLACK FLORETTE have been successfully implemented in font faces other than Zapf Dingbats with glyph shapes that are similar, but not identical to the ITC Zapf Dingbats series 100.

The following guidelines are provided for font developers wishing to support this block of characters. Characters showing large sets of contrastive glyph shapes in the Dingbats block, and in particular the various arrow shapes at U+2794..U+27BE, should have glyphs that are closely modeled on the ITC Zapf Dingbats series 100, which are shown as representative glyphs in the code charts. The same applies to the various stars, asterisks, snowflakes, drop-shadowed squares, check marks, and x's, many of which are ornamental and have elaborate names describing their glyphs.

Where the preceding guidelines do not apply, or where dingbats have more generic applicability as symbols, their glyphs do not need to match the representative glyphs in the code charts in every detail.

Ornamental Brackets. The 14 ornamental brackets encoded at U+2768..U+2775 are part of the set of Zapf Dingbats. Although they have always been included in Zapf Dingbats fonts, they were unencoded in PostScript versions of the fonts on some platforms. The Unicode Standard treats these brackets as punctuation characters.

22.9.5 Ornamental Dingbats: U+1F650–U+1F67F

The block contains a variety of text ornaments and ornamental punctuation marks similar to characters encoded in the main Dingbats block. Most of these symbols are encoded for compatibility with Wingdings or Wingdings 2 font usage; a few derive from Webdings.

Many of these dingbats can be classified as fleurons. These constitute variations on the theme represented by the classic hedera or ivy leaf shape encoded as U+2767 ROTATED FLORAL HEART BULLET.

The block also contains stylistic variants of punctuation marks, including numerous styles of ampersands and et-ligatures, quotation marks, and question marks. These characters extend similar sets of stylized punctuation marks in the Dingbats block. All of these stylized ornamental variants are treated as symbols rather than as true punctuation in the standard.

22.9.6 Alchemical Symbols: U+1F700–U+1F77F

Alchemical symbols were first used by Greek, Syriac, and Egyptian writers around the fifth or sixth century CE and were adopted and proliferated by medieval Arabic and European writers. European alchemists, natural philosophers, chemists, and apothecaries developed and used several parallel systems of symbols while retaining many symbols created by Greek, Syriac, and medieval Arabic writers. Alchemical works published in what is best described as a textbook tradition in the seventeenth and eighteenth centuries routinely included tables of symbols that probably served to spread their use. They became obsolete as alchemy gave way to chemistry. Nevertheless, alchemical symbols continue to be used extensively today in scholarly literature, creative works, New Age texts, and in the gaming and graphics industries.

This block contains a core repertoire of symbols recognized and organized into tables by European writers working in the alchemical textbook tradition approximately 1620–1720. This core repertoire includes all symbols found in the vast majority of the alchemical works of major figures such as Newton, Boyle, and Paracelsus. Some of the most common alchemical symbols have multiple meanings, and are encoded in the Miscellaneous Symbols block, where their usage as alchemical symbols is annotated. For example, U+2609 SUN is also an alchemical symbol for gold.

The character names for the alchemical symbols are in English. Their equivalent Latin names, which often were in greater currency during the period of greatest use of these symbols, are provided as aliases in the code charts. Some alchemical names in English directly derive from the Latin name, such as aquafortis and aqua regia, so in a number of cases the English and Latin names are identical.

22.9.7 Mahjong Tiles: U+1F000–U+1F02F

The characters in this block are game symbols representing the set of tiles used to play the popular Chinese game of mahjong. The exact origin of mahjong is unknown, but it has been around since at least the mid-19th century, and its popularity spread to Japan, Britain, and the United States during the early 20th century.

Like other game symbols in the Unicode Standard, the mahjong tile symbols are intended as abstractions of graphical symbols for game pieces used in text. Simplified, iconic representation of mahjong pieces are printed in game manuals and appear in discussion about the game. There is some variation in the exact set of tiles used in different countries, so the Unicode Standard encodes a superset of the graphical symbols for the tiles used in the various local traditions. The main set of tiles consists of three suits with nine numerical tiles each: the Bamboos, the Circles, and the Characters.

Additional tiles include the Dragons, the Winds, the Flowers, and the Seasons. The blank tile symbol is the so-called *white dragon*. Also included is a black tile symbol, which does not represent an actual game tile, but rather indicates a facedown tile, occasionally seen as a symbol in text about playing mahjong.

22.9.8 Domino Tiles: U+1F030–U+1F09F

This block contains a set of graphical symbols for domino tiles. Dominoes is a game which derives from Chinese tile games dating back to the twelfth century.

Domino tile symbols are used for the "double-six" set of tiles, which is the most common set of dominoes and the only one widely attested in manuals and textual discussion using graphical tile symbols.

The domino tile symbols do not represent the domino pieces per se, but instead constitute graphical symbols for particular orientations of the dominoes, because orientation of the tiles is significant in discussion of dominoes play. Each visually distinct rotation of a domino tile is separately encoded. Thus, for example, both U+1F081 DOMINO TILE VERTICAL-04-02 and U+1F04F DOMINO TILE HORIZONTAL-04-02 are encoded, as well as U+1F075 DOMINO TILE VERTICAL-02-04 and U+1F043 DOMINO TILE HORIZONTAL-02-04. All four of those symbols represent the same game tile, but each orientation of the tile is visually distinct and requires its own symbol for text. The digits in the character names for the domino tile symbols reflect the dot patterns on the tiles.

Two symbols do not represent particular tiles of the double-six set of dominoes, but instead are graphical symbols for a domino tile turned facedown.

22.9.9 Playing Cards: U+1F0A0–U+1F0FF

The symbols in this block are used to represent the 52-card deck most commonly used today, and the 56-card deck used in some European games; the latter includes a Knight in addition to Jack, Queen, and King. These cards map completely to the Minor Arcana of the Western Tarot from which they derive, and are unified with the latter. The symbols for trumps in the range U+1F0E0..U+1F0F5 occur as playing cards in some traditional German, Italian, and French decks. These trumps are

historically derived from the 22 Major Arcana of the esoteric Western Tarot sets. The combined set can be used to represent the 78 cards of the common tarot decks.

Also included in this block are a generic card back and three jokers. U+1F0CF PLAYING CARD BLACK JOKER is used in one of the Japanese cell phone carrier emoji sets; its presentation may be in color and need not be black. U+1F0BF PLAYING CARD RED JOKER occurs in some card decks as a third joker.

These characters most commonly appear as the Anglo-French-style playing cards used with international bridge or poker. However, playing card characters may have a variety of different appearances depending on language and usage. In different countries, the suits, colors and numbers may be substantially different, to the point of being unrecognizable. For example, the letters on face cards may vary (English cards use "K" for "king," while French cards use "R" for "roi"); the digits on the numbered cards may appear as a Western "10" or as "१०" in Hindi, and the appearance of the suits may differ (Swiss playing cards depict acorns rather than clubs, while traditional tarot cards use swords rather than spades). The background decoration of cards may also vary radically. When used to represent the cards of divination tarot decks, the visual appearance is usually very different and much more complex.

No one should expect reliable interchange of a particular appearance of the playing card characters without additional information (such as a font) or agreement between sender and receiver. Without such information or agreement, someone viewing an online document may see substantially different glyphs from what the writer intended.

Basic playing card suit symbols are encoded in the Miscellaneous Symbols block in the range U+2660..U+2667.

22.9.10 Chess Symbols: U+1FA00–U+1FA6F

The Chess Symbols block contains extensions for chess notations beyond the basic Western chess symbols found in the Miscellaneous Symbols block. The chess symbols in the range U+1FA00..U+1FA53 are used in a variety of heterodox Western chess notations, also widely referred to as "fairy chess." These notations include the introduction of new or hybrid chess pieces, such as grasshoppers, nightriders, equihoppers, or various blends of knights with other pieces. There are also a number of neutral pieces, which conceptually belong neither to the white side nor the black side, often displayed with one side of the piece black and the other side of the piece shown with an outlined glyph. Many of these symbols simply consist of existing Western chess symbols for orthodox pieces, inverted or turned sideways. This practice dates from the time when printers would often take existing cast metal sorts and physically invert or turn them before locking them into the forme, to create new symbols for printing heterodox chess problems and commentary.

This block also contains a set of circled CJK ideographic symbols used in Chinese chess (*Xiangqi*) notation, in the range U+1FA60..U+1FA6D. These symbols come in separate "red" and "black" sets, abstractly representing the two sets of seven pieces in that game. In actual practice, both for the symbols printed on the pieces in Chinese chess sets and in notation, there is considerable variation in the color of the pieces, and in the particular CJK ideograph within the circle. For example, both traditional and simplified characters occur, and there is some other variation in the choice of the CJK ideograph, as well. Because of this variability in the CJK ideograph used, these symbols are treated differently than most regular circled CJK ideographic symbols in the standard. No compatibility decompositions to CJK unified ideographs are given in the UCD or shown in the code charts.

22.9.11 Yijing Hexagram Symbols: U+4DC0–U+4DFF

Usage of the Yijing Hexagram Symbols in China begins with a text called 《周易》 *Zhou Yi,* ("the Zhou Dynasty classic of change"), said to have originated circa 1000 BCE. This text is now popularly known as the *Yijing*, *I Ching*, or *Book of Changes*. These symbols represent a primary level of notation in this ancient philosophical text, which is traditionally considered the first and most important of the Chinese classics. Today, these symbols appear in many print and electronic publications, produced in Asia and all over the world. The important Chinese character lexicon *Hanyu Da Zidian*, for example, makes use of these symbols in running text. These symbols are semantically distinct written signs associated with specific words. Each of the 64 hexagrams has a unique one- or two-syllable name. Each hexagram name is intimately connected with interpretation of the six lines. Related characters are Monogram and Digram Symbols (U+268A..U+268F), Yijing Trigram Symbols (U+2630..U+2637), and Tai Xuan Jing Symbols (U+1D300..U+1D356).

22.9.12 Tai Xuan Jing Symbols: U+1D300–U+1D35F

Usage of these symbols in China begins with a text called 《太玄經》 *Tai Xuan Jing* (literally, "the exceedingly arcane classic"). Composed by a man named 楊雄 Yang Xiong (53 BCE–18 CE), the first draft of this work was completed in 2 BCE, in the decade before the fall of the Western Han Dynasty. This text is popularly known in the West under several titles, including *The Alternative I Ching* and *The Elemental Changes*. A number of annotated editions of *Tai Xuan Jing* have been published and reprinted in the 2,000 years since the original work appeared.

These symbols represent a primary level of notation in the original ancient text, following and expanding upon the traditions of the Chinese classic *Yijing*. The tetragram signs are less well known and less widely used than the hexagram signs. For this reason they were encoded on Plane 1 rather than the BMP.

Monograms. U+1D300 MONOGRAM FOR EARTH is an extension of the traditional Yijing monogram symbols, U+268A MONOGRAM FOR YANG and U+268B MONO-GRAM FOR YIN. Because *yang* is typically associated with heaven (Chinese *tian*) and *yin* is typically associated with earth (Chinese *di*), the character U+1D300 has an unfortunate name. Tai Xuan Jing studies typically associate it with human (Chinese *ren*), as midway between heaven and earth.

Digrams. The range of characters U+1D301..U+1D302 constitutes an extension of the Yijing digram symbols encoded in the range U+268C..U+268F. They consist of the combinations of the human (*ren*) monogram with either the *yang* or the *yin* monogram. Because of the naming problem for U+1D300, these digrams also have infelicitous character names. Users are advised to identify the digram symbols by their representative glyphs or by the Chinese aliases provided for them in the code charts.

Tetragrams. The bulk of the symbols in the Tai Xuan Jing Symbols block are the tetragram signs. These tetragram symbols are semantically distinct written signs associated with specific words. Each of the 81 tetragrams has a unique monosyllabic name, and each tetragram name is intimately connected with interpretation of the four lines.

The 81 tetragram symbols (U+1D306..U+1D356) encoded on Plane 1 constitute a complete set. Within this set of 81 signs, a subset of 16 signs known as the Yijing tetragrams is of importance to Yijing scholarship. These are used in the study of the "nuclear trigrams." Related characters are the Yijing Trigram symbols (U+2630..U+2637) and the Yijing Hexagram symbols (U+4DC0..U+4DFF).

22.9.13 Ancient Symbols: U+10190–U+101CF

This block contains ancient symbols, none of which are in modern use. Typically, they derive from ancient epigraphic, papyrological, or manuscript traditions, and represent miscellaneous symbols not specifically included in blocks dedicated to particular ancient scripts. The first set of these consists of ancient Roman symbols for weights and measures, and symbols used in Roman coinage.

Similar symbols can be found in the Ancient Greek Numbers block, U+10140..U+1018F.

22.9.14 Phaistos Disc: U+101D0–U+101FF

The Phaistos disc was found during an archaeological dig in Phaistos, Crete about a century ago. The small fired clay disc is imprinted on both sides with a series of symbols, arranged in a spiral pattern. The disc probably dates from the mid-18th to the mid-14th century BCE.

The symbols have not been deciphered, and the disc remains the only known example of these symbols. Because there is nothing to compare them to, and the corpus is so limited, it is not even clear whether the symbols constitute a writing system for a language or are something else entirely. Nonetheless, the disc has engendered great interest, and numerous scholars and amateurs spend time discussing the symbols.

The repertoire of symbols is noncontroversial, as they were incised in the disc by stamping preformed seals into the clay. Most of the symbols are clearly pictographic in form. The entire set is encoded in the Phaistos Disc Symbols block as a set of symbols, with no assumptions about their possible meaning and functions. One combining mark is encoded. It represents a hand-carved mark on the disc, which occurs attached to the final sign of groups of other symbols.

Directionality. Scholarly consensus is that the text of the Phaistos disc was inscribed starting from the outer rim of the disc and going inward toward the center. Because of that layout order and the orientation of the spiral, the disc text can be said to have right-to-left directionality. However, the Phaistos disc symbols have been given a default directionality of strong left-to-right in the Unicode Standard. This choice simplifies text layout of the symbols for researchers and would-be decipherers, who wish to display the symbols in the same order as the surrounding left-to-right text (for example, in the Latin script) used to discuss them. The additional complexity of bidirectional layout and editing would be unwelcome in such contexts.

This choice of directionality properties for the Phaistos disc symbols matches the precedent of the Old Italic script. (See *Section 8.6, Old Italic*.) Early Old Italic inscriptions were often laid out from right to left, but the directionality of the Old Italic script in the Unicode Standard is strong left-to-right, to simplify layout using the modern scholarly conventions for discussion of Old Italic texts.

The glyphs for letters of ancient Mediterranean scripts often show mirroring based on line direction. This behavior is well-known, for example, for archaic Greek when written in boustrophedon. Etruscan also displays glyph mirroring of letters. The choice of representative glyphs for the Phaistos disc symbols is based on this mirroring convention, as well. The symbols on the disc are in a right-to-left line context. However, the symbols are given left-to-right directionality in the Unicode Standard, so the representative glyphs in the code charts are reversed (mirrored) from their appearance on the disc.

22.10 Enclosed and Square

There are a large number of compatibility symbols in the Unicode Standard which consist either of letters or numbers enclosed in some graphic element, or which consist of letters or numbers in a square arrangement. Many of these symbols are derived from legacy East Asian character sets, in which such symbols are commonly encoded as elements.

Enclosed Symbols. Enclosed symbols typically consist of a letter, digit, Katakana syllable, Hangul jamo, or CJK ideograph enclosed in a circle or a square. In some cases the enclosure may consist of a pair of parentheses or tortoise-shell brackets, and the enclosed element may also consist of more than a single letter or digit, as for circled numbers 10 through 50. Occasionally the symbol is shown as white on a black encircling background, in which case the character name typically includes the word NEGATIVE.

Many of the enclosed symbols that come in small, ordered sets—the Latin alphabet, kana, jamo, digits, and Han ideographs one through ten—were originally intended for use in text as numbered bullets for lists. Parenthetical enclosures were in turn developed to mimic typewriter conventions for representing circled letters and digits used as list bullets. This functionality has now largely been supplanted by styles and other markup in rich text contexts, but the enclosed symbols in the Unicode Standard are encoded for interoperability with the legacy East Asian character sets and for the occasional text context where such symbols otherwise occur.

A few of the enclosed symbols have conventional meanings unrelated to the usage of encircled letters and digits as list bullets. In some instances these are distinguished in the standard—often because legacy standards separately encoded them. Thus, for example, U+24B8 Ⓒ CIRCLED LATIN CAPITAL LETTER C is distinct from U+00A9 © COPYRIGHT SIGN, even though the two symbols are similar in appearance. In cases where otherwise generic enclosed symbols have specific conventional meanings, those meanings are called out in the code charts with aliases or other annotations. For example, U+1F157 🅗 NEGATIVE CIRCLED LATIN CAPITAL LETTER H is also a commonly occurring map symbol for "hotel."

Square Symbols. Another convention commonly seen in East Asian character sets is the creation of compound symbols by arranging two, three, four, or even more small-sized letters or syllables into a square shape consistent with the typical rendering footprint of a CJK ideograph. One subset of these consists of square symbols for Latin abbreviations, often for SI and other technical units, such as "km" or "km/h"; these square symbols are mostly derived from Korean legacy standards. Another subset consists of Katakana words for units of measurement, classified ad symbols, and many other similar word elements arranged into a square array; these symbols are derived from Japanese legacy standards. A third major subset consists of Chinese

telegraphic symbols for hours, days, and months, consisting of a digit or sequence of digits next to the CJK ideograph for "hour," "day" or "month."

Source Standards. Major sources for the repertoire of enclosed and square symbols in the Unicode Standard include the Korean national standard, KS X 1001:1998; the Chinese national standard, GB 2312:1980; the Japanese national standards JIS X 0208-1997 and JIS X 0213:2000; and CNS 11643. Others derive from the Japanese television standard, ARIB STD B24, and from various East Asian industry standards, such as the Japanese cell phone carrier emoji sets, or corporate glyph registries.

Allocation. The Unicode Standard includes five blocks allocated for the encoding of various enclosed and square symbols. Each of those blocks is described briefly in the text that follows, to indicate which subsets of these symbols it contains and to highlight any other special considerations that may apply to each block. In addition, there are a number of circled digit and number symbols encoded in the Dingbats block (U+2700..U+27BF). Those circled symbols occur in the ITC Zapf dingbats series 100, and most of them were encoded with other Zapf dingbat symbols, rather than being allocated in the separate blocks for enclosed and square symbols. Finally, a small number of circled symbols from ISO/IEC 8859-1 or other sources can be found in the Latin-1 Supplement block (U+0080..U+00FF) or the Letterlike Symbols block (U+2100..U+214F).

Decomposition. Nearly all of the enclosed and square symbols in the Unicode Standard are considered compatibility characters, encoded for interoperability with other character sets. A significant majority of those are also compatibility decomposable characters, given explicit compatibility decompositions in the Unicode Character Database. The general patterns for these decompositions are described here. For full details for any particular one of these symbols, see the code charts or consult the data files in the UCD.

Parenthesized symbols are decomposed to sequences of opening and closing parentheses surrounding the letter or digit(s) of the symbol. Square symbols consisting of digit(s) followed by a full stop or a comma are decomposed into the digit sequence and the full stop or comma. Square symbols consisting of several Katakana syllables are decomposed into the corresponding sequence of Katakana characters and are given the decomposition tag "<square>". Similar principles apply to square symbols consisting of sequences of Latin letters and symbols. Chinese telegraphic symbols, consisting of sequences of digits and CJK ideographs, are given compatibility decompositions, but do not have the decomposition tag "<square>".

Circled symbols consisting of a single letter or digit surrounded by a simple circular graphic element are given compatibility decompositions with the decomposition tag "<circle>". Circled symbols with more complex graphic styles, including double circled and negative circled symbols, are simply treated as atomic symbols, and are not decomposed. The same pattern is applied to enclosed symbols where the enclo-

sure is a square graphic element instead of a circle, except that the decomposition tag in those cases is "<square>". Occasionally a "circled" symbol that involves a sequence of Latin letters is preferentially represented with an ellipse surrounding the letters, as for U+1F12E ⓦⓩ CIRCLED WZ, the German *Warenzeichen*. Such elliptic shape is considered to be a typographical adaptation of the circle, and does not constitute a distinct decomposition type in the Unicode Standard.

It is important to realize that the decomposition of enclosed symbols in the Unicode Standard does not make them canonical equivalents to letters or digits in sequence with combining enclosing marks such as U+20DD ◌⃝ COMBINING ENCLOSING CIRCLE. The combining enclosing marks are provided in the Unicode Standard to enable the representation of occasional enclosed symbols not otherwise encoded as characters. There is also no defined way of indicating the application of a combining enclosing mark to more than a single base character. Furthermore, full rendering support of the application of enclosing combining marks, even to single base characters, is not widely available. Hence, in most instances, if an enclosed symbol is available in the Unicode Standard as a single encoded character, it is recommended to simply make use of that composed symbol.

Casing. There are special considerations for the casing relationships of enclosed or square symbols involving letters of the Latin alphabet. The *circled* letters of the Latin alphabet come in an uppercase set (U+24B6..U+24CF) and a lowercase set (U+24D0..U+24EA). Largely because the compatibility decompositions for those symbols are to a single letter each, these two sets are given the derived properties, Uppercase and Lowercase, respectively, and case map to each other. The superficially similar *parenthesized* letters of the Latin alphabet also come in an uppercase set (U+1F110..U+1F129) and a lowercase set (U+24BC..U+24B5), but are not case mapped to each other and are not given derived casing properties. This difference is in part because the compatibility decompositions for these parenthesized symbols are to sequences involving parentheses, instead of single letters, and in part because the uppercase set was encoded many years later than the lowercase set. Square symbols consisting of arbitrary sequences of Latin letters, which themselves may be of mixed case, are simply treated as caseless symbols in the Unicode Standard.

22.10.1 Enclosed Alphanumerics: U+2460–U+24FF

The enclosed symbols in this block consist of single Latin letters, digits, or numbers —most enclosed by a circle. The block also contains letters, digits, or numbers enclosed in parentheses, and a series of numbers followed by full stop. All of these symbols are intended to function as numbered (or lettered) bullets in ordered lists, and most are encoded for compatibility with major East Asian character sets.

The circled numbers one through ten (U+2461..U+2469) are also considered to be unified with the comparable set of circled black numbers with serifs on a white back-

ground from the ITC Zapf Dingbats series 100. Those ten symbols are encoded in this block, instead of in the Dingbats block.

The negative circled numbers eleven through twenty (U+24EB..U+24F4) are a continuation of the set of circled white numbers with serifs on a black background, encoded at U+2776..U+277F in the Dingbats block.

22.10.2 Enclosed CJK Letters and Months: U+3200–U+32FF

This block contains large sets of circled or parenthesized Japanese Katakana, Hangul jamo, or CJK ideographs, from East Asian character sets. It also contains circled numbers twenty-one through fifty, which constitute a continuation of the series of circled numbers from the Enclosed Alphanumerics block. There are also a small number of Chinese telegraph symbols and square Latin abbreviations, which are continuations of the larger sets primarily encoded in the CJK Compatibility block.

The enclosed symbols in the range U+3248..U+324F, which consist of circled numbers ten through eighty on white circles centered on black squares, are encoded for compatibility with the Japanese television standard, ARIB STD B24. In that standard, they are intended to represent symbols for speed limit signs, expressed in kilometers per hour.

The Japanese era name, Reiwa (Japanese: 令和), is encoded at U+32FF SQUARE ERA NAME REIWA. The Reiwa era began on May 1, 2019. The prior era, Heisei (Japanese: 平成), began on January 8, 1989 and ended on April 30, 2019. The SQUARE ERA NAME HEISEI and three additional era names are encoded in the range U+337B..U+337E.

22.10.3 CJK Compatibility: U+3300–U+33FF

The CJK Compatibility block consists entirely of square symbols encoded for compatibility with various East Asian character sets. These come in four sets: square Latin abbreviations, Chinese telegraph symbols for hours and days, squared Katakana words, and a small set of Japanese era names.

Squared Katakana words are Katakana-spelled words that fill a single display cell (em-square) when intermixed with CJK ideographs. Likewise, the square Latin abbreviation symbols are designed to fill a single character position when mixed with CJK ideographs. Note that modern software for the East Asian market can often support the comparable functionality via styles that allow typesetting of arbitrary Katakana words or Latin abbreviations in an em-square. Such solutions are preferred when available, as they are not limited to specific lists of encoded symbols such as those in this block.

Japanese Era Names. The Japanese era name symbols refer to the dates given in <u>Table 22-9</u>.

Table 22-9. Japanese Era Names

Code Point	Name	Dates
U+32FF	SQUARE ERA NAME REIWA	2019-05-01 to present day
U+337B	SQUARE ERA NAME HEISEI	1989-01-08 to 2019-04-30
U+337C	SQUARE ERA NAME SYOUWA	1926-12-25 to 1989-01-07
U+337D	SQUARE ERA NAME TAISYOU	1912-07-30 to 1926-12-24
U+337E	SQUARE ERA NAME MEIZI	1868-10-23 to 1912-07-29

22.10.4 Enclosed Alphanumeric Supplement: U+1F100–U+1F1FF

This block contains more enclosed and square symbols based on Latin letters or digits. Many are encoded for compatibility with the Japanese television standard, ARIB STD B24; others are encoded for compatibility with the Japanese cell phone carrier emoji sets.

Regional Indicator Symbols. A set of 26 regional indicator symbols is encoded in the range U+1F1E6..U+1F1FF. These 26 symbols correspond to a set of Latin letters A through Z, but they do not have letter properties and are not cased. They are intended for use in pairs to represent ISO 3166 region codes. This mechanism does not supplant actual ISO 3166 region codes, which simply use Latin letters from the ASCII range. Pairs of regional indicator symbols should not be construed as *being* region codes (or "country codes"); rather, they constitute convenient indexes into a 26 x 26 array whose elements can be associated with region codes for the purposes of identification, processing, and rendering.

The representative glyph for a single regional indicator symbol is just a dotted box containing a capital Latin letter. The Unicode Standard does not prescribe how the pairs of regional indicator symbols should be rendered. However, current industry practice widely interprets pairs of regional indicator symbols as representing a flag associated with the corresponding ISO 3166 region code. This practice is detailed in the separate Unicode Technical Standard #51, "Unicode Emoji." That specification includes data tables that list precisely which pairs are interpreted for any given version of UTS #51. Charts are also available showing representative flag glyphs for these interpreted pairs, displayed as part of the emoji symbol sets for many mobile platforms.

Conformance to the Unicode Standard does not require conformance to UTS #51. However, the interpretation and display of pairs of regional indicator symbols as specified in UTS #51 is now widely deployed, so in practice it is not advisable to attempt to interpret pairs of regional indicator symbols as representing anything other than an emoji flag.

Regional indicator symbols have specialized properties and behavior related to segmentation, which help to keep interpreted pairs together for line breaking, word segmentation, and so forth.

The file EmojiSources.txt in the Unicode Character Database provides more information about source mappings from pairs of regional indicator symbols to flag emoji in older carrier emoji sets. Provision of roundtrip mappings to those flag emoji was the original impetus to include regional indicator symbols in the Unicode Standard.

Creative Commons License Symbols. Creative Commons license symbols are widely used across web platforms, content creation tools, and search engines to describe a variety of functions, permissions, and concepts related to intellectual property. The set of seven symbols was designed to work efficiently on printed pages, web pages, and signage while following the pattern of a graphic form within a circle.

Six of the seven symbols are encoded in two ranges (U+1F10D..U+1F10F and U+1F16D..U+1F16F). One Creative Commons symbol, the circled equals sign, is represented by U+229C CIRCLED EQUALS.

22.10.5 Enclosed Ideographic Supplement: U+1F200–U+1F2FF

This block consists mostly of enclosed ideographic symbols. It also contains some additional squared Katakana word symbols. Most of the symbols in this block are either encoded for compatibility with the Japanese television standard ARIB STD B24, and intended primarily for use in closed captioning, or are encoded for compatibility with the Japanese cell phone carrier emoji sets.

The enclosed ideographic symbols in the range U+1F210..U+1F23B are enclosed in a square, instead of a circle. One subset of these are symbols referring to broadcast terminology, and the other subset are symbols used in baseball in Japan.

The enclosed ideographic symbols in the range U+1F240..U+1F248 are enclosed in tortoise shell brackets, and are also used in baseball scoring in Japan.

The circled ideographic symbols in the range U+1F260..U+1F265 are felicitous symbols commonly associated with Chinese folk religion. Five of these are collectively referred to as the "five-fold happiness," representing luck, prosperity, longevity, happiness, and wealth. The sixth, U+1F264, represents "double-happiness," a doubled variant of the happiness symbol, associated with love and marriage. Each of these symbols is paired with a respective deity in traditional folk religion.

Chapter 23

Special Areas and Format Characters

This chapter describes several kinds of characters that have special properties as well as areas of the codespace that are set aside for special purposes.

The Unicode Standard contains code positions for the 64 control characters and the DEL character found in ISO standards and many vendor character sets. The choice of control function associated with a given character code is outside the scope of the Unicode Standard, with the exception of those control characters specified in this chapter.

Layout controls are not themselves rendered visibly, but influence the behavior of algorithms for line breaking, word breaking, glyph selection, and bidirectional ordering.

Surrogate code points are restricted use. The numeric values for surrogates are used in pairs in UTF-16 to access 1,048,576 supplementary code points in the range U+10000..U+10FFFF.

Variation selectors allow the specification of standardized variants of characters. This ability is particularly useful where the majority of implementations would treat the two variants as two forms of the same character, but where some implementations need to differentiate between the two. By using a variation selector, such differentiation can be made explicit.

Private-use characters are reserved for private use. Their meaning is defined by private agreement.

Noncharacters are code points that are permanently reserved and will never have characters assigned to them.

The Specials block contains characters that are neither graphic characters nor traditional controls.

Tag characters were intended to support a general scheme for the internal tagging of text streams in the absence of other mechanisms, such as markup languages. The use of tag characters for language tagging is deprecated.

23.1 Control Codes

There are 65 code points set aside in the Unicode Standard for compatibility with the C0 and C1 control codes defined in the ISO/IEC 2022 framework. The ranges of these code points are U+0000..U+001F, U+007F, and U+0080..U+009F, which correspond to the 8-bit controls 00_{16} to $1F_{16}$ (C0 controls), $7F_{16}$ (*delete*), and 80_{16} to $9F_{16}$ (C1 controls), respectively. For example, the 8-bit legacy control code *character tabulation* (or *tab*) is the byte value 09_{16}; the Unicode Standard encodes the corresponding control code at U+0009.

The Unicode Standard provides for the intact interchange of these code points, neither adding to nor subtracting from their semantics. The semantics of the control codes are generally determined by the application with which they are used. However, in the absence of specific application uses, they may be interpreted according to the control function semantics specified in ISO/IEC 6429:1992.

In general, the use of control codes constitutes a higher-level protocol and is beyond the scope of the Unicode Standard. For example, the use of ISO/IEC 6429 control sequences for controlling bidirectional formatting would be a legitimate higher-level protocol layered on top of the plain text of the Unicode Standard. Higher-level protocols are not specified by the Unicode Standard; their existence cannot be assumed without a separate agreement between the parties interchanging such data.

23.1.1 Representing Control Sequences

There is a simple, one-to-one mapping between 7-bit (and 8-bit) control codes and the Unicode control codes: every 7-bit (or 8-bit) control code is numerically equal to its corresponding Unicode code point. For example, if the ASCII *line feed* control code ($0A_{16}$) is to be used for line break control, then the text "WX<LF>YZ" would be transmitted in Unicode plain text as the following coded character sequence: <0057, 0058, 000A, 0059, 005A>.

Control sequences that are part of Unicode text must be represented in terms of the Unicode encoding forms. For example, suppose that an application allows embedded font information to be transmitted by means of markup using plain text and control codes. A font tag specified as "^ATimes^B", where ^A refers to the C0 control code 01_{16} and ^B refers to the C0 control code 02_{16}, would then be expressed by the following coded character sequence: <0001, 0054, 0069, 006D, 0065, 0073, 0002>. The representation of the control codes in the three Unicode encoding forms simply follows the rules for any other code points in the standard:

UTF-8: <01 54 69 6D 65 73 02>

UTF-16: <0001 0054 0069 006D 0065 0073 0002>

UTF-32: <00000001 00000054 00000069 0000006D

00000065 00000073 00000002>

Escape Sequences. Escape sequences are a particular type of protocol that consists of the use of some set of ASCII characters introduced by the *escape* control code, $1B_{16}$, to convey extra-textual information. When converting escape sequences into and out of Unicode text, they should be converted on a character-by-character basis. For instance, "ESC-A" <1B 41> would be converted into the Unicode coded character sequence <001B, 0041>. Interpretation of U+0041 as part of the escape sequence, rather than as *latin capital letter a*, is the responsibility of the higher-level protocol that makes use of such escape sequences. This approach allows for low-level conversion processes to conformantly convert escape sequences into and out of the Unicode Standard without needing to actually recognize the escape sequences as such.

If a process uses escape sequences or other configurations of control code sequences to embed additional information about text (such as formatting attributes or structure), then such sequences constitute a higher-level protocol that is outside the scope of the Unicode Standard.

23.1.2 Specification of Control Code Semantics

Several control codes are commonly used in plain text, particularly those involved in line and paragraph formatting. The use of these control codes is widespread and important to interoperability. Therefore, the Unicode Standard specifies semantics for their use with the rest of the encoded characters in the standard. *Table 23-1* lists those control codes.

Table 23-1. Control Codes Specified in the Unicode Standard

Code Point	Abbreviation	ISO/IEC 6429 Name
U+0009	HT	character tabulation (tab)
U+000A	LF	line feed
U+000B	VT	line tabulation (vertical tab)
U+000C	FF	form feed
U+000D	CR	carriage return
U+001C	FS	information separator four
U+001D	GS	information separator three
U+001E	RS	information separator two
U+001F	US	information separator one
U+0085	NEL	next line

The control codes in <u>*Table 23-1*</u> have the Bidi_Class property values of S, B, or WS, rather than the default of BN used for other control codes. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm.") In particular, U+001C..U+001E and U+001F have the Bidi_Class property values B and S, respectively, so that the Bidirectional Algorithm recognizes their separator semantics.

The control codes U+0009..U+000D and U+0085 have the White_Space property. They also have line breaking property values that differ from the default CM value for other control codes. (See Unicode Standard Annex #14, "Unicode Line Breaking Algorithm.")

U+0000 *null* may be used as a Unicode string terminator, as in the C language. Such usage is outside the scope of the Unicode Standard, which does not require any particular formal language representation of a string or any particular usage of null.

Newline Function. In particular, one or more of the control codes U+000A *line feed*, U+000D *carriage return*, and the Unicode equivalent of the EBCDIC *next line* can encode a *newline function*. A newline function can act like a *line separator* or a *paragraph separator*, depending on the application. See <u>*Section 23.2, Layout Controls*</u>, for information on how to interpret a line or paragraph separator. The exact encoding of a newline function depends on the application domain. For information on how to identify a newline function, see <u>*Section 5.8, Newline Guidelines*</u>.

23.2 Layout Controls

The effect of layout controls is specific to particular text processes. As much as possible, layout controls are transparent to those text processes for which they were not intended. In other words, their effects are mutually orthogonal.

23.2.1 Line and Word Breaking

This subsection summarizes the intended behavior of certain layout controls which affect line and word breaking. Line breaking and word breaking are distinct text processes. Although a candidate position for a line break in text often coincides with a candidate position for a word break, there are also many situations where candidate break positions of different types do not coincide. The implications for the interaction of layout controls with text segmentation processes are complex. For a full description of line breaking, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm." For a full description of other text segmentation processes, including word breaking, see Unicode Standard Annex #29, "Unicode Text Segmentation."

No-Break Space. U+00A0 NO-BREAK SPACE has the same width as U+0020 SPACE, but the NO-BREAK SPACE indicates that, under normal circumstances, no line breaks are permitted between it and surrounding characters, unless the preceding or following character is a line or paragraph separator or space or zero width space. For a complete list of space characters in the Unicode Standard, see *Table 6-2*.

Word Joiner. U+2060 WORD JOINER behaves like U+00A0 NO-BREAK SPACE in that it indicates the absence of line breaks; however, the *word joiner* has no width. The function of the character is to indicate that line breaks are not allowed between the adjoining characters, except next to hard line breaks. For example, the *word joiner* can be inserted after the fourth character in the text "base+delta" to indicate that there should be no line break between the "e" and the "+". The *word joiner* can be used to prevent line breaking with other characters that do not have nonbreaking variants, such as U+2009 THIN SPACE or U+2015 HORIZONTAL BAR, by bracketing the character.

The word joiner must not be confused with the *zero width joiner* or the *combining grapheme joiner*, which have very different functions. In particular, inserting a word joiner between two characters has no effect on their ligating and cursive joining behavior. The word joiner should be ignored in contexts other than line breaking. Note in particular that the word joiner is ignored for word *segmentation*. (See Unicode Standard Annex #29, "Unicode Text Segmentation.")

Zero Width No-Break Space. In addition to its primary meaning of *byte order mark* (see "Byte Order Mark" in *Section 23.8, Specials*), the code point U+FEFF possesses

the semantics of ZERO WIDTH NO-BREAK SPACE, which matches that of *word joiner.* Until Unicode 3.2, U+FEFF was the only code point with word joining semantics, but because it is more commonly used as *byte order mark,* the use of U+2060 WORD JOINER to indicate word joining is strongly preferred for any new text. Implementations should continue to support the word joining semantics of U+FEFF for backward compatibility.

Zero Width Space. The U+200B ZERO WIDTH SPACE indicates a word break or line break opportunity, even though there is no intrinsic width associated with this character. Zero-width space characters are intended to be used in languages that have no visible word spacing to represent word break or line break opportunities, such as Thai, Myanmar, Khmer, and Japanese.

The "zero width" in the character name for ZWSP should not be understood too literally. While this character ordinarily does not result in a visible space between characters, text justification algorithms may add inter-character spacing (letter spacing) between characters separated by a ZWSP. For example, in *Table 23-2*, the row labeled "Display 4" illustrates incorrect suppression of inter-character spacing in the context of a ZWSP.

Table 23-2. Letter Spacing

Type	Justification Examples	Comment
Memory	the ISP®[ZWSP]Charts	The [ZWSP] is inserted to allow line break after ®
Display 1	the ISP®Charts	Without letter spacing
Display 2	the I S P ® C h a r t s	Increased letter spacing
Display 3	t h e I S P ® C h a r t s	"Thai-style" letter spacing
Display 4	t h e I S P ®C h a r t s	[ZWSP] incorrectly inhibiting letter spacing (after ®)

This behavior for ZWSP contrasts with that for fixed-width space characters, such as U+2002 EN SPACE. Such spaces have a specified width that is typically unaffected by justification and which should not be increased (or reduced) by inter-character spacing (see *Section 6.2, General Punctuation*).

In some languages such as German and Russian, increased letter spacing is used to indicate emphasis. Implementers should be aware of this issue.

Zero-Width Spaces and Joiner Characters. The zero-width spaces are not to be confused with the zero-width joiner characters. U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER have no effect on word or line break boundaries, and ZERO WIDTH NO-BREAK SPACE and ZERO WIDTH SPACE have no effect on joining

or linking behavior. The zero-width joiner characters should be ignored when deter-
mining word or line break boundaries. See "Cursive Connection" later in this section.

Hyphenation. U+00AD SOFT HYPHEN (SHY) indicates an intraword break point,
where a line break is preferred if a word must be hyphenated or otherwise broken
across lines. Such break points are generally determined by an automatic hyphenator.
SHY can be used with any script, but its use is generally limited to situations where
users need to override the behavior of such a hyphenator. The visible rendering of a
line break at an intraword break point, whether automatically determined or indi-
cated by a SHY, depends on the surrounding characters, the rules governing the
script and language used, and, at times, the meaning of the word. The precise rules
are outside the scope of this standard, but see Unicode Standard Annex #14,
"Unicode Line Breaking Algorithm," for additional information. A common default
rendering is to insert a hyphen before the line break, but this is insufficient or even
incorrect in many situations.

Contrast this usage with U+2027 HYPHENATION POINT, which is used for a visible
indication of the place of hyphenation in dictionaries. For a complete list of dash
characters in the Unicode Standard, including all the hyphens, see *Table 6-3*.

The Unicode Standard includes two nonbreaking hyphen characters: U+2011 NON-
BREAKING HYPHEN and U+0F0C TIBETAN MARK DELIMITER TSHEG BSTAR. See
Section 13.4, Tibetan, for more discussion of the Tibetan-specific line breaking
behavior.

Line and Paragraph Separator. The Unicode Standard provides two unambiguous
characters, U+2028 LINE SEPARATOR and U+2029 PARAGRAPH SEPARATOR, to sepa-
rate lines and paragraphs. They are considered the default form of denoting line and
paragraph boundaries in Unicode plain text. A new line is begun after each LINE
SEPARATOR. A new paragraph is begun after each PARAGRAPH SEPARATOR. As these
characters are separator codes, it is not necessary either to start the first line or para-
graph or to end the last line or paragraph with them. Doing so would indicate that
there was an empty paragraph or line following. The PARAGRAPH SEPARATOR can be
inserted between paragraphs of text. Its use allows the creation of plain text files,
which can be laid out on a different line width at the receiving end. The LINE SEPA-
RATOR can be used to indicate an unconditional end of line.

A paragraph separator indicates where a new paragraph should start. Any interpara-
graph formatting would be applied. This formatting could cause, for example, the
line to be broken, any interparagraph line spacing to be applied, and the first line to
be indented. A *line separator* indicates that a line break should occur at this point;
although the text continues on the next line, it does not start a new paragraph—no
interparagraph line spacing or paragraphic indentation is applied. For more informa-
tion on line separators, see *Section 5.8, Newline Guidelines*.

23.2.2 Cursive Connection and Ligatures

In some fonts for some scripts, consecutive characters in a text stream may be rendered via adjacent glyphs that cursively join to each other, so as to emulate connected handwriting. For example, cursive joining is implemented in nearly all fonts for the Arabic scripts and in a few handwriting-like fonts for the Latin script.

Cursive rendering is implemented by joining glyphs in the font and by using a process that selects the particular joining glyph to represent each individual character occurrence, based on the joining nature of its neighboring characters. This glyph selection is implemented in the rendering engine, typically using information in the font.

In many cases there is an even closer binding, where a sequence of characters is represented by a single glyph, called a ligature. Ligatures can occur in both cursive and noncursive fonts. Where ligatures are available, it is the task of the rendering system to select a ligature to create the most appropriate line layout. However, the rendering system cannot define the locations where ligatures are possible because there are many languages in which ligature formation requires more information. For example, in some languages, ligatures are never formed across syllable boundaries.

On occasion, an author may wish to override the normal automatic selection of connecting glyphs or ligatures. Typically, this choice is made to achieve one of the following effects:

- Cause nondefault joining appearance (for example, as is sometimes required in writing Persian using the Arabic script)
- Exhibit the joining-variant glyphs themselves in isolation
- Request a ligature to be formed where it normally would not be
- Request a ligature not to be formed where it normally would be

The Unicode Standard provides two characters that influence joining and ligature glyph selection: U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER. The zero width joiner and non-joiner request a rendering system to have more or less of a connection between characters than they would otherwise have. Such a connection may be a simple cursive link, or it may include control of ligatures.

The zero width joiner and non-joiner characters are designed for use in plain text; they should not be used where higher-level ligation and cursive control is available. (See the W3C specification, "Unicode in XML and Other Markup Languages," for more information.) Moreover, they are essentially requests for the rendering system

to take into account when laying out the text; while a rendering system should consider them, it is perfectly acceptable for the system to disregard these requests.

The ZWJ and ZWNJ are designed for marking the unusual cases where ligatures or cursive connections are required or prohibited. These characters are not to be used in all cases where ligatures or cursive connections are desired; instead, they are meant only for overriding the normal behavior of the text.

Joiner. U+200D ZERO WIDTH JOINER requests a more connected rendering of adjacent characters. In particular:

- If the two characters could form a ligature but do not normally, ZWJ requests that the ligature be used.

- Otherwise, if either of the characters could cursively connect but do not normally, ZWJ requests that each of the characters take a cursive-connection form where possible.

- A typical use of ZWJ is to show the connected form of a character without a visible neighbor, as shown in *Figure 23-2*.

In a sequence like <X, ZWJ, Y>, where a cursive form exists for X but not for Y, the presence of ZWJ requests a cursive form for X. Otherwise, where neither a ligature nor a cursive connection is available, the ZWJ has no effect. In other words, given the three broad categories below, ZWJ requests that glyphs in the highest available category (for the given font) be used:

1. Ligated
2. Cursively connected
3. Unconnected

Non-joiner. U+200C ZERO WIDTH NON-JOINER is intended to break both cursive connections and ligatures in rendering.

ZWNJ requests that glyphs in the lowest available category (for the given font) be used.

For those unusual circumstances where someone wants to forbid ligatures in a sequence XY but promote cursive connection, the sequence <X, ZWJ, ZWNJ, ZWJ, Y> can be used. The ZWNJ breaks ligatures, while the two adjacent joiners cause the X and Y to take adjacent cursive forms (where they exist). Similarly, if someone wanted to have X take a cursive form but Y be isolated, then the sequence <X, ZWJ, ZWNJ, Y> could be used (as in previous versions of the Unicode Standard). Examples are shown in *Figure 23-3*.

Cursive Connection. For cursive connection, the joiner and non-joiner characters typically do not modify the contextual selection process itself, but instead change the context of a particular character occurrence. By providing a non-joining adjacent character where the adjacent character otherwise would be joining, or vice versa, they indicate that the rendering process should select a different joining glyph. This process can be used in two ways: to prevent a cursive joining or to exhibit joining glyphs in isolation.

In *Figure 23-1*, the insertion of the ZWNJ overrides the normal cursive joining of *sad* and *lam*.

Figure 23-1. Prevention of Joining

In *Figure 23-2*, the normal display of *ghain* without ZWJ before or after it uses the nominal (isolated) glyph form. When preceded and followed by ZWJ characters, however, the *ghain* is rendered with its medial form glyph in isolation.

Figure 23-2. Exhibition of Joining Glyphs in Isolation

The examples in *Figure 23-1* and *Figure 23-2* are adapted from the Iranian national coded character set standard, ISIRI 3342, which defines ZWNJ and ZWJ as "pseudo space" and "pseudo connection," respectively.

Examples. *Figure 23-3* provides samples of desired renderings when the joiner or non-joiner is inserted between two characters. The examples presume that all of the glyphs are available in the font. If, for example, the ligatures are not available, the display would fall back to the unligated forms. Each of the entries in the first column of *Figure 23-3* shows two characters in visual display order. The column headings show characters to be inserted between those two characters. The cells below show the respective display when the joiners in the heading row are inserted between the original two characters.

Figure 23-3. Effect of Intervening Joiners

Character Sequences	As Is	ZWNJ	ZWJ ZWNJ ZWJ	ZWJ
f i 0066 0069	f i *or* fi	f i	f i	fi
ا ل 0627 0644	لا	لا	لا	لا
ج م 062C 0645	مج *or* جم	مج	مج	مج
ج و 062C 0648	وج	وج	وج	وج

For backward compatibility, between Arabic characters a ZWJ acts just like the sequence <ZWJ, ZWNJ, ZWJ>, preventing a ligature from forming instead of requesting the use of a ligature that would not normally be used. As a result, there is no plain text mechanism for requesting the use of a ligature in Arabic text.

Transparency. The property value of Joining_Type = Transparent applies to characters that should not interfere with cursive connection, even when they occur in sequence between two characters that are connected cursively. These include all nonspacing marks and most format control characters, except for ZWJ and ZWNJ themselves. Note, in particular, that enclosing combining marks are also transparent as regards cursive connection. For example, using U+20DD COMBINING ENCLOSING CIRCLE to circle an Arabic letter in a sequence should not cause that Arabic letter to change its cursive connections to neighboring letters. See *Section 9.2, Arabic*, for more on joining classes and the details regarding Arabic cursive joining.

Joiner and Non-joiner in Indic Scripts. In Indic text, the ZWJ and ZWNJ are used to request particular display forms. A ZWJ after a sequence of consonant plus virama requests what is called a "half-form" of that consonant. A ZWNJ after a sequence of consonant plus virama requests that conjunct formation be interrupted, usually resulting in an explicit virama on that consonant. There are a few more specialized uses as well. For more information, see the discussions in *Chapter 12, South and Central Asia-I*.

Implementation Notes. For modern font technologies, such as OpenType or AAT, font vendors should add ZWJ to their ligature mapping tables as appropriate. Thus, where a font had a mapping from "f" + "i" to fi, the font designer should add the mapping from "f" + ZWJ + "i" to fi. In contrast, ZWNJ will normally have the desired effect naturally for most fonts without any change, as it simply obstructs the normal ligature/cursive connection behavior. As with all other alternate format characters, fonts should use an invisible zero-width glyph for representation of both ZWJ and ZWNJ.

Filtering Joiner and Non-joiner. ZERO WIDTH JOINER and ZERO WIDTH NON-JOINER are format control characters. As such, and in common with other format control characters, they are ordinarily ignored by processes that analyze text content. For example, a spell-checker or a search operation should filter them out when checking for matches. There are exceptions, however. In particular scripts—most notably the Indic scripts—ZWJ and ZWNJ have specialized usages that may be of orthographic significance. In those contexts, blind filtering of all instances of ZWJ or ZWNJ may result in ignoring distinctions relevant to the user's notion of text content. Implementers should be aware of these exceptional circumstances, so that searching and matching operations behave as expected for those scripts.

23.2.3 Prepended Concatenation Marks

The Unicode Standard includes a number of format characters that combine with a sequence of one or more characters of a specific category, where the entire sequence is then rendered as a single, ligated display unit. These *prepended concatenation marks* always have a visible display, despite their status as format characters. An example is U+0601 ARABIC SIGN SANAH, the Arabic year sign, which is followed by a sequence of Arabic-Indic digits to format a year. See "Signs Spanning Numbers" in *Section 9.2, Arabic* for more discussion of the use and display of these signs in the Arabic script.

A sequence anchored by a prepended concatenation mark behaves somewhat analogously to a combining character sequence, with the prepended concatenation mark standing in as the "base character" of the sequence. The entire sequence is a tightly defined syntactic element and ends with the first character that is not in the specified category (often a digit) required for that particular mark. Inserting any other character, even a default ignorable code point, will disrupt the unit. The entire sequence constitutes a single unit for display; however, unlike a combining character sequence, the sequence introduced by a prepended concatenation mark does not form a grapheme cluster for the purposes of cursor movement and similar editing processes. The overall behavior more closely resembles that of normal ligatures in text.

Many of the prepended concatenation marks occur in the Arabic script, and require an unbroken sequence of decimal digits in the Arabic script. In other instances, as for U+070F SYRIAC ABBREVIATION MARK, the expectation is that they will be followed by an unbroken sequence of Syriac letters.

U+2044 FRACTION SLASH behaves somewhat similarly to prepended concatenation marks. Although it is categorized as an ordinary math symbol and not as a format character, it is intended to result in a ligated unit fraction (such as ¾ instead of 3/4) when positioned between two unbroken sequences of decimal digits (gc=Nd). As for the character sequence following a prepended concatenation mark, the digit sequences for a fraction slash are interrupted even by a default ignorable code point such as U+2060 WORD JOINER.

23.2.4 Combining Grapheme Joiner

U+034F COMBINING GRAPHEME JOINER (CGJ) is used to affect the collation of adjacent characters for purposes of language-sensitive collation and searching. It is also used to distinguish sequences that would otherwise be canonically equivalent.

Formally, the combining grapheme joiner is not a format control character, but rather a combining mark. It has the General_Category value gc = Mn and the canonical combining class value ccc = 0.

As a result of these properties, the presence of a combining grapheme joiner in the midst of a combining character sequence does not interrupt the combining character sequence; any process that is accumulating and processing all the characters of a combining character sequence would include a combining grapheme joiner as part of that sequence. This differs from the behavior of most format control characters, whose presence would interrupt a combining character sequence.

In addition, because the combining grapheme joiner has the canonical combining class of 0, canonical reordering will not reorder any adjacent combining marks around a combining grapheme joiner. (See the discussion of canonical ordering in *Section 3.11, Normalization Forms*.) In turn, this means that insertion of a combining grapheme joiner between two combining marks will prevent normalization from switching the positions of those two combining marks, regardless of their own combining classes.

Blocking Reordering. The CGJ has no visible glyph and no other format effect on neighboring characters but simply blocks reordering of combining marks. It can therefore be used as a tool to distinguish two alternative orderings of a sequence of combining marks for some exceptional processing or rendering purpose, whenever normalization would otherwise eliminate the distinction between the two sequences.

For example, using CGJ to block reordering is one way to maintain distinction between differently ordered sequences of certain Hebrew accents and marks. These distinctions are necessary for analytic and text representational purposes. However, these characters were assigned fixed-position combining classes despite the fact that they interact typographically. As a result, normalization treats differently ordered sequences as equivalent. In particular, the sequence

    <lamed, patah, hiriq, finalmem>

is canonically equivalent to

    <lamed, hiriq, patah, finalmem>

because the canonical combining classes of U+05B4 HEBREW POINT HIRIQ and U+05B7 HEBREW POINT PATAH are distinct. However, the sequence

<lamed, patah, CGJ, hiriq, finalmem>

is not canonically equivalent to the other two. The presence of the combining grapheme joiner, which has ccc = 0, blocks the reordering of *hiriq* before *patah* by canonical reordering and thus allows a *patah* following a *hiriq* and a *patah* preceding a *hiriq* to be reliably distinguished, whether for display or for other processing.

The use of CGJ with double diacritics is discussed in *Section 7.9, Combining Marks*; see *Figure 7-11*.

CGJ and Collation. The Unicode Collation Algorithm normalizes Unicode text strings before applying collation weighting. The combining grapheme joiner is ordinarily ignored in collation key weighting in the UCA. However, whenever it blocks the reordering of combining marks in a string, it affects the order of secondary key weights associated with those combining marks, giving the two strings distinct keys. That makes it possible to treat them distinctly in searching and sorting without having to tailor the weights for either the combining grapheme joiner or the combining marks.

The CGJ can also be used to prevent the formation of contractions in the Unicode Collation Algorithm. For example, while "ch" is sorted as a single unit in a tailored Slovak collation, the sequence <c, CGJ, h> will sort as a "c" followed by an "h". The CGJ can also be used in German, for example, to distinguish in sorting between "ü" in the meaning of u-umlaut, which is the more common case and often sorted like <u,e>, and "ü" in the meaning u-diaeresis, which is comparatively rare and sorted like "u" with a secondary key weight. This also requires no tailoring of either the combining grapheme joiner or the sequence. Because CGJ is invisible and has the Default_Ignorable_Code_Point property, data that are marked up with a CGJ should not cause problems for other processes.

It is possible to give sequences of characters that include the combining grapheme joiner special tailored weights. Thus the sequence <c, CGJ, h> could be weighted completely differently from the contraction "ch" or from the way "c" and "h" would have sorted without the contraction. However, such an application of CGJ is not recommended. For more information on the use of CGJ with sorting, matching, and searching, see Unicode Technical Report #10, "Unicode Collation Algorithm."

Rendering. For rendering, the combining grapheme joiner is invisible. However, some older implementations may treat a sequence of grapheme clusters linked by combining grapheme joiners as a single unit for the application of enclosing combining marks. For more information on grapheme clusters, see Unicode Technical Report #29, "Unicode Text Segmentation." For more information on enclosing combining marks, see *Section 3.11, Normalization Forms*.

CGJ and Joiner Characters. The combining grapheme joiner must not be confused with the *zero width joiner* or the *word joiner*, which have very different functions. In

particular, inserting a combining grapheme joiner between two characters should have no effect on their ligation or cursive joining behavior. Where the prevention of line breaking is the desired effect, the word joiner should be used. For more information on the behavior of these characters in line breaking, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm."

23.2.5 Bidirectional Ordering Controls

Bidirectional ordering controls are used in the Bidirectional Algorithm, described in Unicode Standard Annex #9, "Unicode Bidirectional Algorithm." Systems that handle right-to-left scripts such as Arabic, Syriac, and Hebrew, for example, should interpret these format control characters. The bidirectional ordering controls are shown in *Table 23-3*.

Table 23-3. Bidirectional Ordering Controls

Code	Name	Abbreviation
U+061C	ARABIC LETTER MARK	ALM
U+200E	LEFT-TO-RIGHT MARK	LRM
U+200F	RIGHT-TO-LEFT MARK	RLM
U+202A	LEFT-TO-RIGHT EMBEDDING	LRE
U+202B	RIGHT-TO-LEFT EMBEDDING	RLE
U+202C	POP DIRECTIONAL FORMATTING	PDF
U+202D	LEFT-TO-RIGHT OVERRIDE	LRO
U+202E	RIGHT-TO-LEFT OVERRIDE	RLO
U+2066	LEFT-TO-RIGHT ISOLATE	LRI
U+2067	RIGHT-TO-LEFT ISOLATE	RLI
U+2068	FIRST STRONG ISOLATE	FSI
U+2069	POP DIRECTIONAL ISOLATE	PDI

As with other format control characters, bidirectional ordering controls affect the layout of the text in which they are contained but should be ignored for other text processes, such as sorting or searching. However, text processes that modify text content must maintain these characters correctly, because matching pairs of bidirectional ordering controls must be coordinated, so as not to disrupt the layout and interpretation of bidirectional text. Each instance of a LRE, RLE, LRO, or RLO is normally paired with a corresponding PDF. Likewise, each instance of an LRI, RLI, or FSI is normally paired with a corresponding PDI.

U+200E LEFT-TO-RIGHT MARK, U+200F RIGHT-TO-LEFT MARK, and U+061C ARABIC LETTER MARK have the semantics of an invisible character of zero width, except that these characters have strong directionality. They are intended to be used to resolve cases of ambiguous directionality in the context of bidirectional texts; they

are not paired. Unlike U+200B ZERO WIDTH SPACE, these characters carry no word breaking semantics. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," for more information.)

23.2.6 Stateful Format Controls

The Unicode Standard contains a small number of *paired stateful controls.* These characters are used in pairs, with an initiating character (or sequence) and a terminating character. Even when these characters are not supported by a particular implementation, complications can arise due to their paired nature. Whenever text is cut, copied, pasted, or deleted, these characters can become unpaired. To avoid this problem, ideally both any copied text and its context (site of a deletion, or target of an insertion) would be modified so as to maintain all pairings that were in effect for each piece of text. This process can be quite complicated, however, and is not often done—or is done incorrectly if attempted.

The paired stateful controls recommended for use are listed in *Table 23-4*.

<div align="center">

Table 23-4. Paired Stateful Controls

</div>

Characters	Documentation
Bidi Overrides, Embeddings, and Isolates	*Section 23.2, Layout Controls*; UAX #9
Annotation Characters	*Section 23.8, Specials*
Musical Beams and Slurs	*Section 21.2, Western Musical Symbols*

The bidirectional overrides, embeddings, and isolates, as well as the annotation characters are reasonably robust, because their behavior terminates at paragraph boundaries. Paired format controls for representation of beams and slurs in music are recommended only for specialized musical layout software, and also have limited scope.

Bidirectional overrides, embeddings, and isolates are default ignorable (that is, Default_Ignorable_Code_Point = True); if they are not supported by an implementation, they should not be rendered with a visible glyph. The paired stateful controls for musical beams and slurs are likewise default ignorable.

The annotation characters, however, are different. When they are used and correctly interpreted by an implementation, they separate annotation text from the annotated text, and the fully rendered text will typically distinguish the two parts quite clearly. Simply omitting any display of the annotation characters by an implementation which does not interpret them would have the potential to cause significant misconstrual of text content. Hence, the annotation characters are not default ignorable; an implementation which does not interpret them should render them with visible glyphs, using one of the techniques discussed in *Section 5.3, Unknown and Missing Characters*. See "Annotation Characters" in *Section 23.8, Specials* for more discussion.

Other paired stateful controls in the standard are deprecated, and their use should be avoided. They are listed in *Table 23-5*.

Table 23-5. Paired Stateful Controls (Deprecated)

Characters	Documentation
Deprecated Format Characters	*Section 23.3, Deprecated Format Characters*
U+E0001 LANGUAGE TAG	*Section 23.9, Tag Characters*

The tag characters, originally intended for the representation of language tags, are particularly fragile under editorial operations that move spans of text around. See *Section 5.10, Language Information in Plain Text*, for more information about language tagging.

23.3 Deprecated Format Characters

23.3.1 Deprecated Format Characters: U+206A–U+206F

Three pairs of deprecated format characters are encoded in this block:

- Symmetric swapping format characters used to control the glyphs that depict characters such as "(" (The default state is *activated.*)
- Character shaping selectors used to control the shaping behavior of the Arabic compatibility characters (The default state is *inhibited.*)
- Numeric shape selectors used to override the normal shapes of the Western digits (The default state is *nominal.*)

The use of these character shaping selectors and codes for digit shapes is *strongly* discouraged in the Unicode Standard. Instead, the appropriate character codes should be used with the default state. For example, if contextual forms for Arabic characters are desired, then the nominal characters should be used, not the presentation forms with the shaping selectors. Similarly, if the Arabic digit forms are desired, then the explicit characters should be used, such as U+0660 ARABIC-INDIC DIGIT ZERO.

Symmetric Swapping. The symmetric swapping format characters are used in conjunction with the class of left- and right-handed pairs of characters (symmetric characters), such as parentheses. The characters thus affected are listed in *Section 4.7, Bidi Mirrored*. They indicate whether the interpretation of the term LEFT or RIGHT in the character names should be interpreted as meaning *opening* or *closing*, respectively. They do not nest. The default state of symmetric swapping may be set by a higher-level protocol or standard, such as ISO 6429. In the absence of such a protocol, the default state is *activated*.

From the point of encountering U+206A INHIBIT SYMMETRIC SWAPPING format character up to a subsequent U+206B ACTIVATE SYMMETRIC SWAPPING (if any), the symmetric characters will be interpreted and rendered as left and right.

From the point of encountering U+206B ACTIVATE SYMMETRIC SWAPPING format character up to a subsequent U+206A INHIBIT SYMMETRIC SWAPPING (if any), the symmetric characters will be interpreted and rendered as opening and closing. This state (*activated*) is the default state in the absence of any symmetric swapping code or a higher-level protocol.

Character Shaping Selectors. The character shaping selector format characters are used in conjunction with Arabic presentation forms. During the presentation process, certain letterforms may be joined together in cursive connection or ligatures. The shaping selector codes indicate that the character shape determination (glyph

selection) process used to achieve this presentation effect is to be either activated or inhibited. The shaping selector codes do not nest.

From the point of encountering a U+206C INHIBIT ARABIC FORM SHAPING format character up to a subsequent U+206D ACTIVATE ARABIC FORM SHAPING (if any), the character shaping determination process should be inhibited. If the backing store contains Arabic presentation forms (for example, U+FE80..U+FEFC), then these forms should be presented without shape modification. This state (*inhibited*) is the default state in the absence of any character shaping selector or a higher-level protocol.

From the point of encountering a U+206D ACTIVATE ARABIC FORM SHAPING format character up to a subsequent U+206C INHIBIT ARABIC FORM SHAPING (if any), any Arabic presentation forms that appear in the backing store should be presented with shape modification by means of the character shaping (glyph selection) process.

The shaping selectors have no effect on nominal Arabic characters (U+0660..U+06FF), which are always subject to character shaping (glyph selection).

Numeric Shape Selectors. The numeric shape selector format characters allow the selection of the shapes in which the digits U+0030..U+0039 are to be rendered. These format characters do not nest.

From the point of encountering a U+206E NATIONAL DIGIT SHAPES format character up to a subsequent U+206F NOMINAL DIGIT SHAPES (if any), the European digits (U+0030..U+0039) should be depicted using the appropriate national digit shapes as specified by means of appropriate agreements. For example, they could be displayed with shapes such as the Arabic-Indic digits (U+0660..U+0669). The actual character shapes (glyphs) used to display national digit shapes are not specified by the Unicode Standard.

From the point of encountering a U+206F NOMINAL DIGIT SHAPES format character up to a subsequent U+206E NATIONAL DIGIT SHAPES (if any), the European digits (U+0030..U+0039) should be depicted using glyphs that represent the nominal digit shapes shown in the code tables for these digits. This state (*nominal*) is the default state in the absence of any numeric shape selector or a higher-level protocol.

23.4 Variation Selectors

Characters in the Unicode Standard can be represented by a wide variety of glyphs, as discussed in *Chapter 2, General Structure*. Occasionally the need arises in text processing to restrict or change the set of glyphs that are to be used to represent a character. Normally such changes are indicated by choice of font or style in rich text documents. In special circumstances, such a variation from the normal range of appearance needs to be expressed side-by-side in the same document in plain text contexts, where it is impossible or inconvenient to exchange formatted text. For example, in languages employing the Mongolian script, sometimes a specific variant range of glyphs is needed for a specific textual purpose for which the range of "generic" glyphs is considered inappropriate.

Variation selectors provide a mechanism for specifying a restriction on the set of glyphs that are used to represent a particular character. They also provide a mechanism for specifying variants, such as for CJK ideographs and Mongolian letters, that have essentially the same semantics but substantially different ranges of glyphs.

Variation Sequence. A variation sequence is a two-character sequence in which a variation selector follows an initial character. Each variation sequence defines a variant of the initial character. The initial character must have the following properties:

- *D50*: Graphic character (gc=L, M, N, P, S, Zs)
- Not a Variation_Selector
- ccc=0 (does not reorder)
- NFD_QC=Yes (does not decompose)
- NFC_QC=Yes (does not get consumed in composition)

For the definition of NFD_QC and NFC_QC, see Section 5.7.5, "Decompositions and Normalization," in Unicode Standard Annex #44, "Unicode Character Database."

For example, the following types of characters cannot be the initial character of a variation sequence: Control codes, format control characters, most diacritics, some Indic dependent vowels, most viramas, Hangul Jamo medial vowels, and canonical composite characters.

These constraints are required because it is important that variation sequences remain stable under normalization, and that the effect of a variation selector can always be characterized as unambiguously applying to a single character. Versions of the Unicode Standard prior to Version 16.0 had a more limited statement of constraints on variation sequences.

In a variation sequence the variation selector affects the appearance of the initial character. Such changes in appearance may, in turn, have a visual impact on subsequent characters, particularly combining characters applied to that initial character. For example, if the initial character changes shape, that should result in a corresponding change in shape or position of applied combining marks. If the initial character changes color, as can be the case for emoji variation sequences, the color may also change for applied combining marks. If the initial character changes in advance width, that would also change the positioning of subsequent spacing characters.

In particular, the emoji variation sequences for digits, U+0023 "#" NUMBER SIGN, and U+002A "*" ASTERISK are intended to affect the color, size, and positioning of U+20E3 ☐ COMBINING ENCLOSING KEYCAP when applied to those initial characters. For example, the variation sequence <0023, FE0F> selects the emoji presentation variant for "#". The sequence <0023, FE0F, 20E3> should show the *enclosing keycap* with an appropriate emoji style, matching the "#" in color, shape, and positioning. Shape changes for variation sequences, with or without additional combining marks, may also result in an increase of advance width; thus, each of the sequences <0023, FE0F>, <0023, 20E3>, and <0023, FE0F, 20E3> may have a distinct advance width, differing from U+0023 alone.

The use of variation selectors is *not* intended as a general extension mechanism for the character encoding. Combinations of particular initial characters plus particular variation selectors have no effect on display unless they occur in pre-defined lists maintained by the Unicode Consortium. The three sanctioned lists are as follows:

> *Standardized variation sequences* are defined in the file StandardizedVariants.txt in the Unicode Character Database.

> *Emoji variation sequences* are defined in the file emoji-variation-sequences.txt, associated with Unicode Technical Standard #51, "Unicode Emoji."

> *Ideographic variation sequences* are defined by the registration process defined in Unicode Technical Standard #37, "Unicode Ideographic Variation Database," and are listed in the Ideographic Variation Database.

Only those three types of variation sequences are sanctioned for use by conformant implementations. In all other cases, use of a variation selector character does not change the visual appearance of the preceding initial character from what it would have had in the absence of the variation selector.

The variation selectors themselves are combining marks of combining class 0 and are default ignorable. Thus, if the variation sequence is not supported, the variation selector should be invisible and ignored. This does not preclude modes or environments where the variation selectors should be given visible appearance. For example, a "Show Hidden" mode could reveal the presence of such characters with specialized glyphs, or a particular environment could use or require a visual indication of an

initial character (such as a wavy underline) to show that it is part of a standardized variation sequence that cannot be supported by the current font.

The standardization or support of a particular variation sequence does *not* limit the set of glyphs that can be used to represent the initial character alone. If a user *requires* a visual distinction between a character and a particular variant of that character, then fonts must be used to make that distinction. The existence of a variation sequence does not preclude the later encoding of a new character with distinct semantics and a similar or overlapping range of glyphs.

CJK Compatibility Ideographs. There are 1,002 standardized variation sequences for CJK compatibility ideographs. One sequence is defined for each CJK compatibility ideograph in the Unicode Standard. These sequences are defined to address a normalization issue for these ideographs.

Implementations or users sometimes need a CJK compatibility ideograph to be distinct from its corresponding CJK unified ideograph. For example, a distinct glyphic form may be expected for a particular text. However, CJK compatibility ideographs have canonical equivalence mappings to their corresponding CJK unified ideograph, which means that such distinctions are lost whenever Unicode normalization is applied. Using the variation sequence preserves the distinction found in the original, non-normalized text, even when normalization is later applied.

Because variation sequences are not affected by Unicode normalization, an implementation which uses the corresponding standardized variation sequence can safely maintain the intended distinction for that CJK compatibility ideograph, even in plain text.

It is important to distinguish standardized variation sequences for CJK compatibility ideographs from the variation sequences that are registered in the Ideographic Variation Database (IVD). The former are normalization-stable representations of the CJK compatibility ideographs; they are defined in StandardizedVariants.txt, and there is precisely one variation sequence for each CJK compatibility ideograph. The latter are also stable under normalization, but correspond to implementation-specific glyphs in a registry entry.

Representative Glyphs for Variants. Representative glyphs for most of the standardized variation sequences are included directly in the code charts. See "Standardized Variation Sequences" in *Section 24.1, Character Names List* for an explanation of the conventions used to identify such sequences in the code charts. Emoji variation sequences, which often require large, colorful glyphs for their representation, can be found instead in the emoji charts. See *Appendix B.3, Other Unicode Online Resources*.

Representative glyphs for ideographic variation sequences are located in the pertinent registrations associated with the Ideographic Variation Database.

Mongolian. For the behavior of older implementations of Mongolian using variation selectors, see the discussion of Mongolian free variation selectors in *Section 13.5, Mongolian*.

23.5 Private-Use Characters

Private-use characters are assigned Unicode code points whose interpretation is not specified by this standard and whose use may be determined by private agreement among cooperating users. These characters are designated for private use and do not have defined, interpretable semantics except by private agreement.

Private-use characters are often used to implement end-user defined characters (EUDC), which are common in East Asian computing environments.

No charts are provided for private-use characters, as any such characters are, by their very nature, defined only outside the context of this standard.

Three distinct blocks of private-use characters are provided in the Unicode Standard: the primary Private Use Area (PUA) in the BMP and two supplementary Private Use Areas in the supplemental planes.

All code points in the blocks of private-use characters in the Unicode Standard are permanently designated for private use. No assignment to a particular standard set of characters will ever be endorsed or documented by the Unicode Consortium for any of these code points.

Any prior use of a character as a private-use character has no direct bearing on any eventual encoding decisions regarding whether and how to encode that character. Standardization of characters must always follow the normal process for encoding of new characters or scripts.

Properties. No private agreement can change which character codes are reserved for private use. However, many Unicode algorithms use the General_Category property or properties which are derived by reference to the General_Category property. Private agreements may override the General_Category or derivations based on it, except where overriding is expressly disallowed in the conformance statement for a specific algorithm. In other words, private agreements may define which private-use characters should be treated like spaces, digits, letters, punctuation, and so on, by all parties to those private agreements. In particular, when a private agreement overrides the General_Category of a private-use character from the default value of gc = Co to some other value such as gc = Lu or gc = Nd, such a change does not change its inherent identity as a private-use character, but merely specifies its intended behavior according to the private agreement.

For all other properties the Unicode Character Database also provides default values for private-use characters. Except for normalization-related properties, these default property values should be considered informative. They are intended to allow implementations to treat private-use characters in a consistent way, even in the absence of a particular private agreement, and to simplify the use of common types of private-

use characters. Those default values are based on typical use-cases for private-use characters. Implementations may freely change or override the default values according to their requirements for private use. For example, a private agreement might specify that two private-use characters are to be treated as a case mapping pair, or a private agreement could specify that a private-use character is to be rendered and otherwise treated as a combining mark.

To exchange private-use characters in a semantically consistent way, users may also exchange privately defined data which describes how each private-use character is to be interpreted. The Unicode Standard provides no predefined format for such a data exchange.

Normalization. The canonical and compatibility decompositions of any private-use character are equal to the character itself (for example, U+E000 decomposes to U+E000). The Canonical_Combining_Class of private-use characters is defined as 0 (Not_Reordered). These values are normatively defined by the Unicode Standard and cannot be changed by private agreement. The treatment of all private-use characters for normalization forms NFC, NFD, NFKD, and NFKC is also normatively defined by the Unicode Standard on the basis of these decompositions. (See Unicode Standard Annex #15, "Unicode Normalization Forms.") No private agreement may change these forms—for example, by changing the standard canonical or compatibility decompositions for private-use characters. The implication is that all private-use characters, no matter what private agreements they are subject to, always normalize to themselves and are never reordered in any Unicode normalization form.

This does not preclude private agreements on other transformations. Thus one could define a transformation "MyCompanyComposition" that was identical to NFC except that it mapped U+E000 to "a". The forms NFC, NFD, NFKD, and NFKC themselves, however, cannot be changed by such agreements.

23.5.1 Private Use Area: U+E000–U+F8FF

The primary Private Use Area consists of code points in the range U+E000 to U+F8FF, for a total of 6,400 private-use characters.

Encoding Structure. By convention, the primary Private Use Area is divided into a corporate use subarea for platform writers, starting at U+F8FF and extending downward in values, and an end-user subarea, starting at U+E000 and extending upward.

By following this convention, the likelihood of collision between private-use characters defined by platform writers with private-use characters defined by end users can be reduced. However, it should be noted that this is only a convention, not a normative specification. In principle, any user can define any interpretation of any private-use character.

Corporate Use Subarea. Systems vendors and/or software developers may need to reserve some private-use characters for internal use by their software. The corporate use subarea is the preferred area for such reservations. Assignments of character semantics in this subarea may be completely internal, hidden from end users, and used only for vendor-specific application support, or they may be published as vendor-specific character assignments available to applications and end users. An example of the former case would be the assignment of a character code to a system support operation such as <MOVE> or <COPY>; an example of the latter case would be the assignment of a character code to a vendor-specific logo character such as Apple's *apple* character.

Note, however, that systems vendors may need to support full end-user definability for all private-use characters, for such purposes as *gaiji* support or for transient cross-mapping tables. The use of noncharacters (see *Section 23.7, Noncharacters*, and Definition D14 in *Section 3.4, Characters and Encoding*) is the preferred way to make use of *non-interchangeable* internal system sentinels of various sorts.

End-User Subarea. The end-user subarea is intended for private-use character definitions by end users or for scratch allocations of character space by end-user applications.

Allocation of Subareas. Vendors may choose to reserve ranges of private-use characters in the corporate use subarea and make some defined portion of the end-user subarea available for completely free end-user definition. The convention of separating the two subareas is merely a suggestion for the convenience of system vendors and software developers. No firm dividing line between the two subareas is defined in this standard, as different users may have different requirements. No provision is made in the Unicode Standard for avoiding a "stack-heap collision" between the two subareas; in other words, there is no guarantee that end users will not define a private-use character at a code point that overlaps and conflicts with a particular corporate private-use definition at the same code point. Avoiding such overlaps in definition is up to implementations and users.

23.5.2 Supplementary Private Use Areas

Encoding Structure. The entire Plane 15, with the exception of the noncharacters U+FFFFE and U+FFFFF, is defined to be the Supplementary Private Use Area-A. The entire Plane 16, with the exception of the noncharacters U+10FFFE and U+10FFFF, is defined to be the Supplementary Private Use Area-B. Together these areas make an additional 131,068 code points available for private use.

The supplementary PUAs provide additional undifferentiated space for private-use characters for implementations for which the 6,400 private-use characters in the primary PUA prove to be insufficient.

23.6 Surrogates Area

23.6.1 Surrogates Area: U+D800–U+DFFF

When using UTF-16 to represent supplementary characters, pairs of 16-bit code units are used for each character. These units are called *surrogates*. To distinguish them from ordinary characters, they are allocated in a separate area. The Surrogates Area consists of 1,024 low-half surrogate code points and 1,024 high-half surrogate code points. For the formal definition of a *surrogate pair* and the role of surrogate pairs in the Unicode Conformance Clause, see *Section 3.8, Surrogates,* and *Section 5.4, Handling Surrogate Pairs in UTF-16*.

The use of surrogate pairs in the Unicode Standard is formally equivalent to the Universal Transformation Format-16 (UTF-16) defined in ISO/IEC 10646. For more information, see *Appendix C, Relationship to ISO/IEC 10646*. For a complete statement of UTF-16, see *Section 3.9, Unicode Encoding Forms*.

High-Surrogate. The high-surrogate code points are assigned to the range U+D800..U+DBFF. The high-surrogate code point is always the first element of a surrogate pair.

Low-Surrogate. The low-surrogate code points are assigned to the range U+DC00..U+DFFF. The low-surrogate code point is always the second element of a surrogate pair.

Private-Use High-Surrogates. The high-surrogate code points from U+DB80..U+DBFF are private-use high-surrogate code points (a total of 128 code points). Characters represented by means of a surrogate pair, where the high-surrogate code point is a private-use high-surrogate, are private-use characters from the supplementary private use areas. For more information on private-use characters, see *Section 23.5, Private-Use Characters*.

The code tables do not have charts or name list entries for the range U+D800..U+DFFF because individual, unpaired surrogates merely have code points.

23.7 Noncharacters

23.7.1 Noncharacters: U+FFFE, U+FFFF, and Others

Noncharacters are code points that are permanently reserved in the Unicode Standard for internal use. They are not recommended for use in open interchange of Unicode text data. See *Section 3.2, Conformance Requirements* and *Section 3.4, Characters and Encoding*, for the formal definition of noncharacters and conformance requirements related to their use.

The Unicode Standard sets aside 66 noncharacter code points. The last two code points of each plane are noncharacters: U+FFFE and U+FFFF on the BMP, U+1FFFE and U+1FFFF on Plane 1, and so on, up to U+10FFFE and U+10FFFF on Plane 16, for a total of 34 code points. In addition, there is a contiguous range of another 32 noncharacter code points in the BMP: U+FDD0..U+FDEF. For historical reasons, the range U+FDD0..U+FDEF is contained within the Arabic Presentation Forms-A block, but those noncharacters are not "Arabic noncharacters" or "right-to-left noncharacters," and are not distinguished in any other way from the other noncharacters, except in their code point values.

Applications are free to use any of these noncharacter code points internally. They have no standard interpretation when exchanged outside the context of internal use. However, they are not illegal in interchange, nor does their presence cause Unicode text to be ill-formed. The intent of noncharacters is that they are permanently prohibited from being assigned interchangeable *meanings* by the Unicode Standard. They are not prohibited from occurring in valid Unicode strings which happen to be interchanged. This distinction, which might be seen as too finely drawn, ensures that noncharacters are correctly preserved when "interchanged" internally, as when used in strings in APIs, in other interprocess protocols, or when stored.

If a noncharacter is received in open interchange, an application is not required to interpret it in any way. It is good practice, however, to recognize it as a noncharacter and to take appropriate action, such as replacing it with U+FFFD REPLACEMENT CHARACTER, to indicate the problem in the text. It is not recommended to simply delete noncharacter code points from such text, because of the potential security issues caused by deleting uninterpreted characters. (See conformance clause C7 in *Section 3.2, Conformance Requirements*, and Unicode Technical Report #36, "Unicode Security Considerations.")

In effect, noncharacters can be thought of as application-internal private-use code points. Unlike the private-use characters discussed in *Section 23.5, Private-Use Characters*, which *are* assigned characters and which *are* intended for use in open interchange, subject to interpretation by private agreement, noncharacters are perma-

nently reserved (unassigned) and have no interpretation whatsoever outside of their possible application-internal private uses.

U+FFFF and U+10FFFF. These two noncharacter code points have the attribute of being associated with the largest code unit values for particular Unicode encoding forms. In UTF-16, U+FFFF is associated with the largest 16-bit code unit value, $FFFF_{16}$. U+10FFFF is associated with the largest legal UTF-32 32-bit code unit value, $10FFFF_{16}$. This attribute renders these two noncharacter code points useful for internal purposes as sentinels. For example, they might be used to indicate the end of a list, to represent a value in an index guaranteed to be higher than any valid character value, and so on.

U+FFFE. This noncharacter has the intended peculiarity that, when represented in UTF-16 and then serialized, it has the opposite byte sequence of U+FEFF, the *byte order mark*. This means that applications should reserve U+FFFE as an internal signal that a UTF-16 text stream is in a reversed byte format. Detection of U+FFFE at the start of an input stream should be taken as a strong indication that the input stream should be byte-swapped before interpretation. For more on the use of the *byte order mark* and its interaction with the noncharacter U+FFFE, see <u>*Section 23.8, Specials*</u>.

23.8 Specials

The Specials block contains code points that are interpreted as neither control nor graphic characters but that are provided to facilitate current software practices.

For information about the noncharacter code points U+FFFE and U+FFFF, see *Section 23.7, Noncharacters*.

23.8.1 Byte Order Mark (BOM): U+FEFF

For historical reasons, the character U+FEFF used for the *byte order mark* is named ZERO WIDTH NO-BREAK SPACE. Except for compatibility with versions of Unicode prior to Version 3.2, U+FEFF is not used with the semantics of *zero width no-break space* (see *Section 23.2, Layout Controls*). Instead, its most common and most important usage is in the following two circumstances:

1. Unmarked Byte Order. Some machine architectures use the so-called big-endian byte order, while others use the little-endian byte order. When Unicode text is serialized into bytes, the bytes can go in either order, depending on the architecture. Sometimes this byte order is not externally marked, which causes problems in interchange between different systems.

2. Unmarked Character Set. In some circumstances, the character set information for a stream of coded characters (such as a file) is not available. The only information available is that the stream contains text, but the precise character set is not known.

In these two cases, the character U+FEFF is used as a signature to indicate the byte order and the character set by using the byte serializations described in *Section 3.10, Unicode Encoding Schemes*. Because the byte-swapped version U+FFFE is a noncharacter, when an interpreting process finds U+FFFE as the first character, it signals either that the process has encountered text that is of the incorrect byte order or that the file is not valid Unicode text.

In the UTF-16 encoding scheme, U+FEFF at the very beginning of a file or stream explicitly signals the byte order.

The byte sequences $<FE_{16}\ FF_{16}>$ or $<FF_{16}\ FE_{16}>$ may also serve as a signature to identify a file as containing UTF-16 text. Either sequence is exceedingly rare at the outset of text files using other character encodings, whether single- or multiple-byte, and therefore not likely to be confused with real text data. For example, in systems that employ ISO Latin-1 (ISO/IEC 8859-1) or the Microsoft Windows ANSI Code Page 1252, the byte sequence $<FE_{16}\ FF_{16}>$ constitutes the string *<thorn, y diaeresis>* "þÿ"; in systems that employ the Apple Macintosh Roman character set or the Adobe Stan-

dard Encoding, this sequence represents the sequence *<ogonek, hacek>* "˛ˇ"; in systems that employ other common IBM PC code pages (for example, CP 437, 850), this sequence represents *<black square, no-break space>* "■ ".

In UTF-8, the BOM corresponds to the byte sequence $<EF_{16}\ BB_{16}\ BF_{16}>$. Although there are never any questions of byte order with UTF-8 text, this sequence can serve as signature for UTF-8 encoded text where the character set is unmarked. As with a BOM in UTF-16, this sequence of bytes will be extremely rare at the beginning of text files in other character encodings. For example, in systems that employ Microsoft Windows ANSI Code Page 1252, $<EF_{16}\ BB_{16}\ BF_{16}>$ corresponds to the sequence *<i diaeresis, guillemet, inverted question mark>* "ï » ¿".

For compatibility with versions of the Unicode Standard prior to Version 3.2, the code point U+FEFF has the word-joining semantics of *zero width no-break space* when it is not used as a BOM. In new text, these semantics should be encoded by U+2060 WORD JOINER. See "Line and Word Breaking" in *Section 23.2, Layout Controls*, for more information.

Where the byte order is explicitly specified, such as in UTF-16BE or UTF-16LE, then all U+FEFF characters—even at the very beginning of the text—are to be interpreted as *zero width no-break spaces*. Similarly, where Unicode text has known byte order, initial U+FEFF characters are not required, but for backward compatibility are to be interpreted as *zero width no-break spaces*. For example, for strings in an API, the memory architecture of the processor provides the explicit byte order. For databases and similar structures, it is much more efficient and robust to use a uniform byte order for the same field (if not the entire database), thereby avoiding use of the *byte order mark*.

Systems that use the *byte order mark* must recognize when an initial U+FEFF signals the byte order. In those cases, it is not part of the textual content and should be removed before processing, because otherwise it may be mistaken for a legitimate *zero width no-break space*. To represent an initial U+FEFF ZERO WIDTH NO-BREAK SPACE in a UTF-16 file, use U+FEFF twice in a row. The first one is a *byte order mark*; the second one is the initial *zero width no-break space*. See *Table 23-6* for a summary of encoding scheme signatures.

Table 23-6. Unicode Encoding Scheme Signatures

Encoding Scheme	Signature
UTF-8	EF BB BF
UTF-16 Big-endian	FE FF
UTF-16 Little-endian	FF FE
UTF-32 Big-endian	00 00 FE FF
UTF-32 Little-endian	FF FE 00 00

If U+FEFF had only the semantics of a signature code point, it could be freely deleted from text without affecting the interpretation of the rest of the text. Carelessly appending files together, for example, can result in a signature code point in the middle of text. Unfortunately, U+FEFF also has significance as a character. As a *zero width no-break space*, it indicates that line breaks are not allowed between the adjoining characters. Thus U+FEFF affects the interpretation of text and cannot be freely deleted. The overloading of semantics for this code point has caused problems for programs and protocols. The new character U+2060 WORD JOINER has the same semantics in all cases as U+FEFF, except that it *cannot* be used as a signature. Implementers are strongly encouraged to use *word joiner* in those circumstances whenever word joining semantics are intended.

U+FEFF as Signature in Other Charsets. An initial U+FEFF also takes a characteristic form in other charsets designed for Unicode text. (The term "charset" refers to a wide range of text encodings, including encoding schemes as well as compression schemes and text-specific transformation formats.) The characteristic sequences of bytes associated with an initial U+FEFF can serve as signatures in those cases, as shown in .

Table 23-7. U+FEFF Signature in Other Charsets

Charset	Signature
SCSU	0E FE FF
BOCU-1	FB EE 28
UTF-7	2B 2F 76 38 or 2B 2F 76 39 or 2B 2F 76 2B or 2B 2F 76 2F
UTF-EBCDIC	DD 73 66 73

Most signatures can be deleted either before or after conversion of an input stream into a Unicode encoding form. However, in the case of BOCU-1 and UTF-7, the input byte sequence must be converted before the initial U+FEFF can be deleted, because stripping the signature byte sequence without conversion destroys context necessary for the correct interpretation of subsequent bytes in the input sequence.

Guidelines for Use of a BOM in UTF-8. The UTF-8 encoding scheme permits, but does not require, a BOM to be present. This raises the question of when a BOM should or should not be generated or expected when producing or consuming UTF-8 encoded text.

The utility of a BOM in UTF-8 is limited to scenarios in which a byte sequence contains text that may or may not be encoded as UTF-8. In such scenarios, a BOM

may be useful to differentiate text encoded in one of a few possible character encodings. However, the presence of a BOM may also complicate text processing.

- Some text processing tools fail to handle BOMs correctly. This is especially true for programs that were historically encoding agnostic and for ad hoc programs written for one-time use purposes.

- A text processing tool must maintain additional state in order to recognize if an observed U+FEFF character is a BOM or whether it should be treated as a *zero width no-break space*. Such state may or may not be intrinsic to the structure of the program.

- A text generating tool may be required to generate a BOM if the first character to be encoded is U+FEFF and that character is not intended to be used as a BOM. This is only required for compatibility with Unicode versions prior to 3.2; U+2060 WORD JOINER should be used in place of U+FEFF for such purposes with more recent Unicode versions.

- Concatenation of text containing a BOM requires care. When concatenating to an empty text, preservation of a BOM may be warranted, but otherwise, failure to elide the BOM will result in the insertion of a U+FEFF character that becomes part of the concatenated textual content.

- In situations where text is known to be encoded as UTF-8, a BOM consumes storage space unnecessarily. While this is unlikely to be a concern for a single document, it may be a significant concern in situations involving thousands or millions of small text sources.

The following guidelines advise alternative approaches tailored for a few distinct audiences. Except where otherwise noted, these approaches preclude the possibility of a text starting with a U+FEFF character that is not intended as a BOM under the expectation that such text is exceedingly rare and most likely due to a failure to elide a BOM. Text authored for Unicode 3.2 or later should use U+2060 WORD JOINER instead.

Software developers:

- If consuming UTF-8, recognize and discard a BOM.

- If producing UTF-8, include a BOM only if explicitly directed to do so, or if a BOM is known to be required by a protocol.

Text authors:

- Do not use U+FEFF to function as a *zero width no-break space* character; use U+2060 WORD JOINER instead.

- Include a BOM if one is known to be required by a targeted protocol.

- Otherwise, include a BOM when authoring a UTF-8 text file that contains non-ASCII characters, is not targeting a specific protocol, but which may be opened by applications that will not assume UTF-8 by default. (This is useful on systems like Microsoft Windows where some applications assume text files to be encoded with the Active Code Page.)

- Otherwise, do not include a BOM.

23.8.2 Specials: U+FFF0–U+FFFF

The nine unassigned Unicode code points in the range U+FFF0..U+FFF8 are reserved for special character definitions.

23.8.3 Annotation Characters: U+FFF9–U+FFFB

An *interlinear annotation* consists of *annotating text* that is related to a sequence of *annotated* characters. For all regular editing and text-processing algorithms, the annotated characters are treated as part of the text stream. The annotating text is also part of the content, but for all or some text processing, it does not form part of the main text stream. However, within the annotating text, characters are accessible to the same kind of layout, text-processing, and editing algorithms as the base text. The *annotation character*s delimit the annotating and the annotated text, and identify them as part of an annotation. See *Figure 23-4*.

Figure 23-4. Annotation Characters
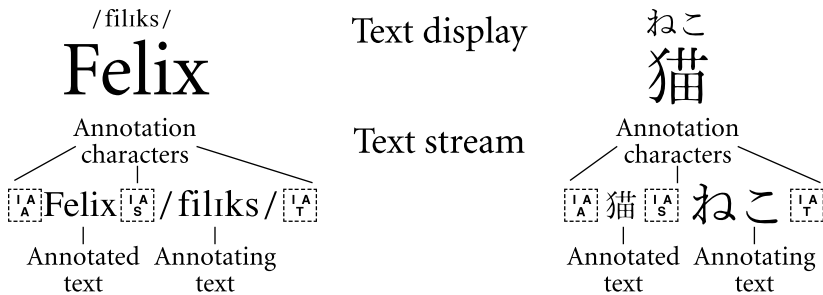
The annotation characters are used in internal processing when out-of-band information is associated with a character stream, very similarly to the usage of U+FFFC OBJECT REPLACEMENT CHARACTER. However, unlike the opaque objects hidden by the latter character, the annotation itself is textual.

Conformance. A conformant implementation that supports annotation characters interprets the base text as if it were part of an unannotated text stream. Within the annotating text, it interprets the annotating characters with their regular Unicode semantics.

U+FFF9 INTERLINEAR ANNOTATION ANCHOR is an anchor character, preceding the interlinear annotation. The exact nature and formatting of the annotation depend on additional information that is not part of the plain text stream. This situation is analogous to that for U+FFFC OBJECT REPLACEMENT CHARACTER.

U+FFFA INTERLINEAR ANNOTATION SEPARATOR separates the base characters in the text stream from the annotation characters that follow. The exact interpretation of this character depends on the nature of the annotation. More than one separator may be present. Additional separators delimit parts of a multipart annotating text.

U+FFFB INTERLINEAR ANNOTATION TERMINATOR terminates the annotation object (and returns to the regular text stream).

Use in Plain Text. Usage of the annotation characters in plain text interchange is strongly discouraged without prior agreement between the sender and the receiver, because the content may be misinterpreted otherwise. Simply filtering out the annotation characters on input will produce an unreadable result or, even worse, an opposite meaning. On input, a plain text receiver should either preserve all characters or remove the interlinear annotation characters as well as the annotating text included between the INTERLINEAR ANNOTATION SEPARATOR and the INTERLINEAR ANNOTATION TERMINATOR.

When an output for plain text usage is desired but the receiver is unknown to the sender, these interlinear annotation characters should be removed as well as the annotating text included between the INTERLINEAR ANNOTATION SEPARATOR and the INTERLINEAR ANNOTATION TERMINATOR.

This restriction does not preclude the use of annotation characters in plain text interchange, but it requires a prior agreement between the sender and the receiver for correct interpretation of the annotations.

Lexical Restrictions. If an implementation encounters a paragraph break between an *anchor* and its corresponding *terminator,* it shall terminate any open annotations at this point. Anchorcharacters must precede their corresponding terminator characters. Unpaired anchors or terminators shall be ignored. A *separator* occurring outside a pair of delimiters, shall be ignored. Annotations may be nested.

Formatting. All formatting information for an annotation is provided by higher-level protocols. The details of the layout of the annotation are implementation-defined. Correct formatting may require additional information that is not present in the character stream, but rather is maintained out-of-band. Therefore, annotation markers serve as placeholders for an implementation that has access to that information from another source. The formatting of annotations and other special line layout features of Japanese is discussed in JIS X 4051.

Input. Annotation characters are not normally input or edited directly by end users. Their insertion and management in text are typically handled by an application, which will present a user interface for selecting and annotating text.

Collation. With the exception of the special case where the annotation is intended to be used as a sort key, annotations are typically ignored for collation or optionally preprocessed to act as tie breakers only. Importantly, annotation base characters are not ignored, but rather are treated like regular text.

Bidirectional Text. Bidirectional processing of text containing interlinear annotations requires special care. This follows from the fact that interlinear annotations are fundamentally nonlinear—the annotations are not part of the main text flow, whereas bidirectional text processing assumes that it is applied to a single, linear text flow. For best results, the Bidirectional Algorithm should be applied to the main text, in which any interlinear annotations are replaced by their annotated text, in each case bracketed by bidirectional format control characters to ensure that the annotated text remains visually contiguous, and then should be separately applied to each extracted segment of annotating text. (See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm," for more information.)

23.8.4 Replacement Characters: U+FFFC–U+FFFD

U+FFFC. The U+FFFC OBJECT REPLACEMENT CHARACTER is used as an insertion point for objects located within a stream of text. All other information about the object is kept outside the character data stream. Internally it is a dummy character that acts as an anchor point for the object's formatting information. In addition to assuring correct placement of an object in a data stream, the object replacement character allows the use of general stream-based algorithms for any textual aspects of embedded objects.

U+FFFD. The U+FFFD REPLACEMENT CHARACTER is the general substitute character in the Unicode Standard. It can be substituted for any "unknown" character in another encoding that cannot be mapped in terms of known Unicode characters. It can also be used as one means of indicating a conversion error, when encountering an ill-formed sequence in a conversion between Unicode encoding forms. See *Section 3.9, Unicode Encoding Forms* for detailed recommendations on the use of U+FFFD as replacement for ill-formed sequences. See also *Section 5.3, Unknown and Missing Characters* for related topics.

23.9 Tag Characters

23.9.1 Tag Characters: U+E0000–U+E007F

This block encodes a set of 97 special-use tag characters to enable the construction of tags using characters that can be strictly separated from ordinary text content characters in Unicode, but that correspond to ASCII-based strings. They can be identified and/or ignored by implementations with trivial algorithms because there is no overloading of usage for these tag characters—they can express only tag values and never textual content itself. One of these 97 characters is the deprecated language tag identification character, U+E0001 LANGUAGE TAG.

The current conformant use of the undeprecated 96 tag characters is specified in Unicode Technical Standard #51, "Unicode Emoji." See *ED-14a. emoji tag sequence* (ETS) and *Annex C, Valid Emoji Tag Sequences* in that specification. UTS #51 does not use nested tag sequences, and the tag sequences it does use are not stateful.

23.9.2 Deprecated Use for Language Tagging

The tag characters were originally intended for language tagging of plain text, as an alternative to using malformed UTF-8 for language tagging. When it became clear that language tagging using these characters was complicated and unnecessary, the tag characters were deprecated as of Version 5.1. In Version 8.0, all but U+E0001 LANGUAGE TAG and U+E007F CANCEL TAG were un-deprecated, in anticipation of other use for the tag characters. In Version 9.0, U+E007F CANCEL TAG was also un-deprecated, and the tag characters were repurposed for use in emoji tag sequences.

For the full specification of the original use of tag characters, see *Section 16.9, Tag Characters* in Version 5.0 of *The Unicode Standard*:

Chapter 24

About the Code Charts

Disclaimer

Character images shown in the code charts are not prescriptive. In actual fonts, considerable variations are to be expected.

The Unicode code charts present the characters of the Unicode Standard. This chapter explains the conventions used in the code charts and provides other useful information about the accompanying names lists.

Characters are organized into related groups called *blocks* (see D10b in *Section 3.4, Characters and Encoding*). Many scripts are fully contained within a single block, but other scripts, including some of the most widely used scripts, have characters divided across several blocks. Separate blocks contain common punctuation characters and different types of symbols.

A character names list follows the code chart for each block. The character names list itemizes every character in that block and provides supplementary information in many cases. A full list of the character names and associated annotations, formatted as a text file, NamesList.txt, is available in the Unicode Character Database. That text file contains syntax conventions which are used by the tooling that formats the PDF versions of the code charts and character names lists. For the full specification of those conventions, see NamesList.html in the Unicode Character Database.

An index to distinctive character names can also be found on the Unicode website.

For information about access to the code charts, the character name index, and the roadmap for future allocations, see *Appendix B.3, Other Unicode Online Resources*.

24.1 Character Names List

The following illustration exemplifies common components found in entries in the character names list. These and other components are described in more detail in the remainder of this section.

(code)	*(image)*	*(entry)*	
00AE	®	REGISTERED SIGN	
		= registered trade mark sign (1.0)	*(Version 1.0 name)*
00AF	¯	MACRON	*(Unicode name)*
		= overline, APL overbar	*(alternative names)*
		• this is a spacing character	*(informative note)*
		→ 02C9 ˉ modifier letter macron	*(cross reference)*
		→ 0304 ̄ combining macron	
		→ 0305 ̅ combining overline	
		≈ 0020 ␠ 0304 ̄	*(compatibility decomposition)*
00E5	å	LATIN SMALL LETTER A WITH RING ABOVE	
		• Danish, Norwegian, Swedish, Walloon	*(sample of language use)*
		≡ 0061 a 030A ̊	*(canonical decomposition)*
228A	⊊	SUBSET OF WITH NOT EQUAL TO	
		~ 228A FE00 ⊊ with stroke through bottom members	*(standardized variation sequence)*

24.1.1 Images in the Code Charts and Character Lists

Each character in these code charts is shown with a representative glyph. A representative glyph is not a prescriptive form of the character, but rather one that enables recognition of the intended character to a knowledgeable user and facilitates lookup of the character in the code charts. In many cases, there are more or less well-established alternative glyphic representations for the same character.

Designers of high-quality fonts will do their own research into the preferred glyphic appearance of Unicode characters. In addition, many scripts require context-dependent glyph shaping, glyph positioning, or ligatures, none of which is shown in the code charts. The Unicode Standard contains many characters that are used in writing minority languages or that are historical characters, often used primarily in manuscripts or inscriptions. Where there is no strong tradition of printed materials, the typography of a character may not be settled. Because of these factors, the glyph image chosen as the representative glyph in these code charts should not be considered a definitive guide to best practice for typographical design.

Fonts. The representative glyphs for the Latin, Greek, and Cyrillic scripts in the code charts are based on a serifed, Times-like font. For non-European scripts, typical typefaces were selected that allow as much distinction as possible among the different characters.

The fonts used for other scripts are similar to Times in that each represents a common, widely used design, with variable stroke width and serifs or similar devices, where applicable, to show each character as distinctly as possible. Sans-serif fonts with uniform stroke width tend to have less visibly distinct characters. In the code charts, sans-serif fonts are used for archaic scripts that predate the invention of serifs, for example.

Alternative Forms. Some characters have alternative forms. For example, even the ASCII character U+0061 LATIN SMALL LETTER A has two common alternative forms: the "a" used in Times and the "ɑ" that occurs in many other font styles. In a Times-like font, the character U+03A5 GREEK CAPITAL LETTER UPSILON looks like "Y"; the form ϒ is common in other font styles.

A different case is U+010F LATIN SMALL LETTER D WITH CARON, which is commonly typeset as ďinstead of ď. In such cases, the code charts show the more common variant in preference to a more didactic archetypical shape.

Many characters have been unified and have different appearances in different language contexts. The shape shown for U+2116 № NUMERO SIGN is a fullwidth shape as it would be used in East Asian fonts. In Cyrillic usage, № is the universally recognized glyph. See *Figure 22-2*.

In certain cases, characters need to be represented by more or less condensed, shifted, or distorted glyphs to make them fit the format of the code charts. For example, U+0D10 ഐ MALAYALAM LETTER AI is shown in a reduced size to fit the character cell.

When characters are used in context, the surrounding text gives important clues as to identity, size, and positioning. In the code charts, these clues are absent. For example, U+2075 ⁵ SUPERSCRIPT FIVE is shown much smaller than it would be in a Times-like text font.

Whenever a more obvious choice for representative glyph may be insufficient to aid in the proper identification of the encoded character, a more distinct variant has been selected as representative glyph instead.

Orientation. Representative glyphs for character in the code charts are oriented as they would normally appear in text with the exception of scripts which are predominantly laid out in vertical lines, as for Mongolian and Phags-pa. Commercial production fonts show Mongolian glyphs with their images turned 90 degrees counterclockwise, which is the appropriate orientation for Mongolian text that is laid out horizontally, such as for embedding in horizontally formatted, left-to-right Chinese text. For normal vertical display of Mongolian text, layout engines typically lay out horizontally, and then rotate the formatted text 90 degrees clockwise. Starting with Unicode 7.0, the code charts display Mongolian glyphs in their horizontal orientation,

following the conventions of commercial Mongolian fonts. Glyphs in the Phags-pa code chart are treated similarly.

24.1.2 Special Characters and Code Points

The code charts and character lists use a number of notational conventions for the representation of special characters and code points. Some of these conventions indicate those code points which are *not* assigned to encoded characters, or are permanently reserved. Other conventions convey information about the type of character encoded, or provide a possible fallback rendering for non-printing characters.

Combining Characters. Combining characters are shown with a dotted circle. This dotted circle is not part of the representative glyph and it would not ordinarily be included as part of any actual glyph for that character in a font. Instead, the relative position of the dotted circle indicates an approximate location of the base character in relation to the combining mark.

093F	ि	DEVANAGARI VOWEL SIGN I
		• stands to the left of the consonant
0940	ी	DEVANAGARI VOWEL SIGN II
0941	ु	DEVANAGARI VOWEL SIGN U

The detailed rules for placement of combining characters with respect to various base characters are implemented by the selected font in conjunction with the rendering system.

During rendering, additional adjustments are necessary. Accents such as U+0302 COMBINING CIRCUMFLEX ACCENT are adjusted vertically and horizontally based on the height and width of the base character, as in " î " versus "Ŵ".

If the display of a combining mark with a dotted circle is desired, U+25CC ◌ DOTTED CIRCLE is often chosen as the base character for the mark.

Dashed Box Convention. There are a number of characters in the Unicode Standard which in normal text rendering have no visible display, or whose only effect is to modify the display of *other* characters in proximity to them. Examples include space characters, control characters, and format characters.

To make such characters easily recognizable and distinguishable in the code charts and in any discussion about the characters, they are represented by a square dashed box. This box surrounds a short mnemonic abbreviation of the character's name. For control codes which do not have a listed abbreviation to serve as a mnemonic, the representative glyph shows XXX inside the dashed box as a placeholder.

0020	SP	SPACE

  • sometimes considered a control code
  • other space characters: 2000 [NO SP] – 200A [H SP]

Where such characters have a typical visual appearance in some contexts, an additional representative image may be used, either alone or with a mnemonic abbreviation.

00AD   [SHY]   SOFT HYPHEN
           = discretionary hyphen
           • commonly abbreviated as SHY

In a few cases of very wide punctuation characters that do not naturally fit into a code chart cell, the representative glyph may be shown with an artificially narrow shape, displayed inside the dashed box, with or without additional annotation, to indicate this adjustment of shape.

2E3A   [2M]   TWO-EM DASH
           = omission dash
           • may be used in Chinese for abrupt change of thought, inserting new content, or continuation of tone or sound
           → 2014 —— em dash

This convention is also used for some graphic characters which are only distinguished by special behavior from another character of the same appearance, or which are subject to unusual rendering requirements.

2011   [NB]   NON-BREAKING HYPHEN
           → 002D - hyphen-minus
           → 00AD [SHY] soft hyphen
           ≈ <noBreak> 2010 ‑ non-breaking hyphen

0D4E   [˙]   MALAYALAM LETTER DOT REPH
           • not used in reformed modern Malayalam orthography

The dashed box convention also applies to the glyphs of combining characters which have no visible display of their own, such as variation selectors (see *Section 23.4, Variation Selectors*).

FE00   [VS 1]   VARIATION SELECTOR-1
           • these are abbreviated VS1, and so on

Sometimes, the combining status of the character is indicated by including a dotted circle inside the dashed box, for example for *viramas* that are intended to be invisible themselves, but which create the conjunct forms of adjacent consonants.

17D2   [◌]   KHMER SIGN COENG

> • functions to indicate that the following Khmer letter is to be rendered
> subscripted
> • shape shown is arbitrary and is not visibly rendered

Even though the presence of the dashed box in the code charts indicates that a character is likely to be a space character, a control character, a format character, or a combining character, it cannot be used to infer the actual General_Category value of that character.

Reserved Characters. Character codes that are marked "<reserved>" are unassigned and reserved for future encoding. Reserved codes are indicated by a ▨ glyph. To ensure readability, many instances of reserved characters have been suppressed from the names list. Reserved codes may also have cross references to assigned characters located elsewhere.

2073   ▨   <reserved>
          → 00B3 ³ superscript three

Noncharacters. Character codes that are marked "<not a character>" refer to noncharacters. They are designated code points that will never be assigned to a character. These codes are indicated by a ■ glyph. Noncharacters are shown in the code charts only where they occur together with other characters in the same block. For a complete list of noncharacters, see *Section 23.7, Noncharacters*.

FFFF   ■   <not a character>

Deprecated Characters. Deprecated characters are characters whose use is strongly discouraged, but which are retained in the standard indefinitely so that existing data remain well defined and can be correctly interpreted. (See D13 in *Section 3.4, Characters and Encoding*.) Deprecated characters are explicitly indicated in the Unicode code charts using annotations or subheads.

24.1.3 Character Names

The character names in the code charts precisely match the normative character names in the Unicode Character Database. Character names are unique and stable. By convention, they are in uppercase. For more information on character names, see *Section 4.8, Name*.

24.1.4 Informative Aliases

An informative alias is an informal, alternate name for a character. Aliases are provided to assist in the correct identification of characters, in some cases providing more commonly known names than the normative character name used in the standard. For example:

002E   .   FULL STOP
          = period, dot, decimal point

Informative aliases are indicated with a "=" in the names list, and by convention are shown in lowercase, except when they include a proper name. (Note that a "=" in the names list may also introduce a *normative* alias, which is distinguished from an informative alias by being shown in uppercase. See the following discussion of normative aliases.)

Multiple aliases for a character may be given in a single informative alias line, in which case each alias is separated by a comma. In other cases, multiple informative alias lines may appear in a single entry. Informative aliases can be used to indicate distinct functions that a character may have; this is particularly common for symbols. For example:

2206   ∆   INCREMENT
          = Laplace operator
          = forward difference
          = symmetric difference of sets

In some complex cases involving many informative aliases, rather than introduce a separate line for each set of related aliases, an informative alias line may also separate groups of aliases with semicolons:

1F70A   ⊹   ALCHEMICAL SYMBOL FOR VINEGAR
          = crucible; acid; distill; atrament; vitriol; red sulfur; borax; wine; alkali salt;
          mercurius vivus, quick silver

Informative aliases for different characters are not guaranteed to be unique. They are maintained editorially, and may be changed, added to, or even be deleted in future versions of the standard, as information accumulates about particular characters and their uses.

Informative aliases may serve as useful alternate choices for identifying characters in user interfaces. The formal character names in the standard may differ in unexpected ways from the more commonly used names for the characters. For example:

00B6   ¶   PILCROW SIGN
          = paragraph sign

Unicode 1.0 Names. Some character names from T*he Unicode Standard, Version 1.0* are indicated in the names list. These are provided only for their historical interest. Where they occur, they also are introduced with a "=" and are shown in lowercase. In addition they are explicitly annotated with a following "1.0" in parentheses. For example:

01C3   !   LATIN LETTER RETROFLEX CLICK
    = latin letter exclamation mark (1.0)

If a Unicode 1.0 name and one or more other informative aliases occurs in a single entry, the Unicode 1.0 name will be given first. For example:

00A6   ¦   BROKEN BAR
    = broken vertical bar (1.0)
    = parted rule (in typography)

Note that informative aliases other than Unicode 1.0 names may also contain clarifying annotations in parentheses.

Jamo Short Names. In the Hangul Jamo block, U+1100..U+11FF, the normative jamo short names from Jamo.txt in the UCD are displayed for convenience of reference. These are also indicated with a "=" in the names list and are shown in uppercase to imply their normative status. For example:

1101   ᄁ   HANGUL CHOSEONG SSANGKIYEOK
    = GG

The Jamo short names do not actually have the status of alternate names; instead they are simply string values associated with the jamo characters, for use by the Unicode Hangul Syllable Name Generation algorithm. See *Section 3.12, Conjoining Jamo Behavior*.

24.1.5 Normative Aliases

A normative character name alias is a formal, unique, and stable alternate name for a character. In limited circumstances, characters are given normative character name aliases where there is a defect in the character name. These normative aliases do not replace the character name, but rather allow users to refer formally to the character without requiring the use of a defective name. For more information, see *Section 4.8, Name*.

Normative aliases which provide information about corrections to defective character names or which provide alternate names in wide use for a Unicode format character are printed in the character names list, preceded by a special symbol ※.

FE18   ⏝   PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRAKCET
    ※ PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRACKET
    • misspelling of "BRACKET" in character name is a known defect
    ≈ <vertical> 3017 ⟧ presentation form for vertical right white lenticular brakcet

Normative aliases serving other purposes, if listed, are shown by convention in all caps, following an "=". In contrast, informative aliases are shown in lowercase. Normative aliases of type "control" typically represent names of control functions as listed in the latest edition of ISO 6429. Normative aliases of type "figment" for control codes are not listed. Normative aliases which represent commonly used abbreviations for control codes or format characters are shown in all caps, enclosed in parentheses. For editorial presentation in the names list, those parenthetical listings may occur on the same lines as informative aliases. See NameAliases.txt in the UCD for the definitive listing of all normative aliases, also including their types, suitable for machine parsing.

24.1.6 Cross References

Cross references (preceded by →) are used to indicate a related character of interest, but without indicating the exact nature of the relation. Cross references are most commonly used to indicate a different character of similar or occasionally identical appearance, which might be confused with the character in question. Cross references are also used to indicate characters with similar names or functions, but with distinct appearances. Cross references may also be used to show linguistic relationships, such as letters used for transliteration in a different script. Some blocks start with a list of cross references that simply point to related characters of interest in other blocks. Examples of various types of cross references follow.

Explicit Inequality. The cross reference indicates that two (or more) characters are not identical, although the representative glyphs that depict them are identical or very close in appearance.

003A   :   COLON
        • also used to denote division or scale; for that mathematical use 2236 : is preferred
        → 0589 : armenian full stop
        → 05C3 : hebrew punctuation sof pasuq
        → 2236 : ratio
        → A789 : modifier letter colon

Related Functions. The cross reference indicates that two (or more) characters have similar functions, although the representative glyphs are distinct. See, for example, the cross references to DIVISION SLASH, DIVIDES, and RATIO in the names list entry for U+00F7 DIVISION SIGN:

00F7   ÷   DIVISION SIGN

= obelus
• occasionally used as an alternate, more visually distinct version of 2212 −
or 2011 ⏎ in some contexts
• historically used as a punctuation mark to denote questionable passages
in manuscripts
→ 070B ⏑ syriac harklean obelus
→ 2052 ⁒ commercial minus sign
→ 2212 − minus sign
→ 2215 ∕ division slash
→ 2223 ∣ divides
→ 2236 ∶ ratio
→ 2797 ➗ heavy division sign

In addition to related mathematical functions, cross references may show other related functions, such as use of distinct symbols in different phonetic transcription systems to represent the same sounds. For example, the cross reference to U+0296 in the following entry shows the IPA equivalent for U+01C1:

01C1   ‖   LATIN LETTER LATERAL CLICK

= double pipe
• Khoisan tradition
• "x" in Zulu orthography
→ 0296 ʖ latin letter inverted glottal stop
→ 2225 ‖ parallel to

Related Names. The cross reference indicates that two (or more) characters have similar and possibly confusable names, although their appearance is distinct.

1F32B   ≋   FOG

→ 1F301 🌁 foggy

Transliteration. The cross reference indicates a character from another script commonly used for transliteration of the character in question. Note that this use of cross references is deliberately limited to a few special cases such as Mongolian:

182E   ᠮ   MONGOLIAN LETTER MA

→ 043C м cyrillic small letter em

This use of cross references is also seen for compatibility digraph letters for Serbo-Croatian:

01C9   lj   LATIN SMALL LETTER LJ

→ 0459 љ cyrillic small letter lje

Blind Cross References. The cross reference notation is also used to point to related characters in other blocks. In these cases, the cross reference is not from any particular code point. For example, the list of cross references at the top of the Currency

Symbols block points to many other currency signs scattered throughout the standard.

In a few instances, a cross reference points from a reserved, unassigned code point. These cross references occur in cases where the structure of a chart might lead a user to expect a particular character at a code point, but the character to use is actually encoded elsewhere. This occurs, for example, in several Indic blocks to point to the shared *danda* characters:

For viram punctuation, use the generic Indic 0964 and 0965.

0A64   ▧   <reserved>
                → 0964 ǀ devanagari danda

0A65   ▧   <reserved>
                → 0965 ǁ devanagari double danda

Cross references are neither exhaustive nor symmetric. Typically a general character would have cross references to more specialized characters, but not the other way around.

24.1.7 Information About Languages

An informative note may include a list of one or more of the languages using that character where this information is considered useful. For case pairs, the annotation is given only for the lowercase form to avoid needless repetition. An ellipsis "..." indicates that the listed languages cited are merely the principal ones among many.

24.1.8 Case Mappings

When a case mapping corresponds *solely* to a difference based on SMALL versus CAPITAL in the names of the characters, the case mapping is not given in the names list but only in the Unicode Character Database.

0041   A   LATIN CAPITAL LETTER A
01F2   Dz   LATIN CAPITAL LETTER D WITH SMALL LETTER Z
                ≈ 0044 D 007A z

When the case mapping cannot be predicted from the name, the casing information is sometimes given in a note.

00DF   ß   LATIN SMALL LETTER SHARP S

    = Eszett
    • German
    • not used in Swiss High German
    • uppercase is "SS" or 1E9E ẞ
    • typographically the glyph for this character can be based on a ligature of
    017F ſ with either 0073 s or with an old-style glyph for 007A z (the latter
    similar in appearance to 0292 ʒ). Both forms exist interchangeably today.
    → 03B2 β greek small letter beta

For more information about case and case mappings, see *Section 4.2, Case*.

24.1.9 Decompositions

The decomposition sequence (one or more letters) given for a character is either its
canonical mapping or its compatibility mapping. The canonical mapping is marked
with an *identical to* symbol ≡.

00E5   å   LATIN SMALL LETTER A WITH RING ABOVE
             • Danish, Norwegian, Swedish, Walloon
             ≡ 0061 a 030A ̊
212B   Å   ANGSTROM SIGN
             ≡ 00C5 Å angstrom sign

Compatibility mappings are marked with an *almost equal to* symbol ≈. Formatting
information may be indicated with a formatting tag, shown inside angle brackets.

01F2   Dz   LATIN CAPITAL LETTER D WITH SMALL LETTER Z
             ≈ 0044 D 007A z
FF21   A   FULLWIDTH LATIN CAPITAL LETTER A
             ≈ <wide> 0041 A fullwidth latin capital letter a

The following compatibility formatting tags are used in the Unicode Character Data-
base:

      <font>   A font variant (for example, a blackletter form)

      <noBreak> A no-break version of a space, hyphen, or other punctuation

      <initial>   An initial presentation form (Arabic)

      <medial>   A medial presentation form (Arabic)

      <final>   A final presentation form (Arabic)

      <isolated> An isolated presentation form (Arabic)

      <circle>   An encircled form

      <super>   A superscript form

      <sub>   A subscript form

    <vertical> A vertical layout presentation form

    <wide>   A fullwidth (or zenkaku) compatibility character

    <narrow> A halfwidth (or hankaku) compatibility character

    <small>   A small variant form (CNS compatibility)

    <square>   A CJK squared font variant

    <fraction> A vulgar fraction form

    <compat> Otherwise unspecified compatibility character

In the character names list accompanying the code charts, the "<compat>" label is suppressed, but all other compatibility formatting tags are explicitly listed in the compatibility mapping.

Decomposition mappings are not necessarily full decompositions. For example, the decomposition for U+212B Å ᴀɴɢsᴛʀᴏᴍ sɪɢɴ can be further decomposed using the canonical mapping for U+00C5 Å ʟᴀᴛɪɴ ᴄᴀᴘɪᴛᴀʟ ʟᴇᴛᴛᴇʀ ᴀ ᴡɪᴛʜ ʀɪɴɢ ᴀʙᴏᴠᴇ. (For more information on decomposition, see *Section 3.7, Decomposition*.)

Compatibility decompositions do not attempt to retain or emulate the formatting of the original character. For example, compatibility decompositions with the <noBreak> formatting tag do not use U+2060 ᴡᴏʀᴅ ᴊᴏɪɴᴇʀ to emulate nonbreaking behavior; compatibility decompositions with the <circle> formatting tag do not use U+20DD ᴄᴏᴍʙɪɴɪɴɢ ᴇɴᴄʟᴏsɪɴɢ ᴄɪʀᴄʟᴇ; and compatibility decompositions with formatting tags <initial>, <medial>, <final>, or <isolate> for explicit positional forms do not use ZWJor ZWNJ. The one exception is the use of U+2044 ꜰʀᴀᴄᴛɪᴏɴ sʟᴀsʜ to express the <fraction> semantics of compatibility decompositions for vulgar fractions.

24.1.10 Standardized Variation Sequences

The Unicode Standard defines a number of standardized variation sequences. These consist of a single base character followed by a variation selector. Use of a standardized variation sequence allows a user to indicate their preference for a display with a particular glyph or subset of glyphs for the given character.

In the character names list, each variation sequence for standardized variants is listed in the entry for the base character for that sequence. In some cases a character may be associated with multiple variation sequences. A standardized variation sequence is identified in the character names list with an initial swung dash "~".

  228A   ⊊   SUBSET OF WITH NOT EQUAL TO
                  ~ 228A FE00 ⊊ with stroke through bottom members

Characters for which one or more standardized variants have been defined are displayed in the code charts with a special convention: the code chart cell for such characters has a small black triangle in its upper-right corner.

Characters which have one or more positional glyph variants, but no standardized variants have a small white triangle in the upper-right corner of their code chart cell.

Emoji characters participate in additional emoji-specific variation sequences which are not indicated in the code charts. Those sequences are defined in the emoji-variation-sequences.txt data file.

Blocks containing characters for which standardized variation sequences and/or positional glyph variants are shown in the names list also have a separate summary listing at the end of the block, displaying the variants in a large font size. Each entry in these summary listings is shown as follows:
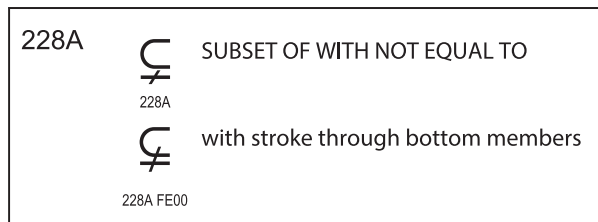
The list of standardized variation sequences in the character names list matches the list defined in the data file StandardizedVariants.txt in the Unicode Character Database. Emoji variation sequences are not included in these summary listings at the ends of blocks, because of the limitations in font technology used for the code chart display. Ideographic variation sequences defined in the Ideographic Variation Database are also not included. See *Section 23.4, Variation Selectors* for more information.

Standardized Variation Sequences to select glyphs appropriate for display of CJK compatibility ideographs are shown not with the corresponding CJK unified ideograph, but rather with the CJK compatibility ideograph defining the glyph to be selected. All CJK compatibility ideographs have a canonical decomposition to a CJK unified ideograph for historical reasons. This means that direct use of CJK compatibility ideographs is problematical, because they are not stable under normalization. To indicate that one of the compatibility glyph shapes is desired, the indicated variation selector can be used with the CJK unified ideograph. In the CJK Compatibility Ideographs and CJK Compatibility Supplement blocks, the canonical decomposition and the relevant standardized variation sequence are shown together with respective

representative glyphs for the sources defined for the CJK compatibility ideograph; see *Figure 24-5*.

Note that there are no indications of variation sequences in the charts for CJK unified ideographs. See the Ideographic Variation Database (IVD) for information on registered variation sequences for CJK unified ideographs.

24.1.11 Emoji Variation Sequences

Many characters with the Emoji property have two associated variation sequences defined in the data file emoji-variation-sequences.txt, one requesting the glyph for text presentation and the other requesting the glyph for emoji presentation. The variation sequences are not listed explicitly in the names list. The glyphs for emoji presentation variation sequences cannot be displayed by the font technology used to produce the code charts. Instead, a representative text presentation is shown throughout. In the code charts, emoji characters that do not have the Emoji_Presentation property and that therefore default to text presentation are indicated with a small *black* triangle in the top left corner:

Emoji characters that also have the Emoji_Presentation property and that therefore would default to that presentation are indicated with a small *white* triangle in the top left corner:

Some characters with the Emoji property also have other variation sequences defined, and so additionally have a small black triangle in the top right corner, as shown in the following example.

Representative glyphs for both the colorful emoji presentation style and the text style of all emoji variation sequences for this version can be found in the emoji charts section of the Unicode website.

24.1.12 Positional Forms

In cursive scripts which have contextually defined positional forms for letters, such as Arabic or Mongolian, the basic positional forms may appear in the code charts. Such forms, when they occur, appear in the charts in the summary listings, together with any standardized variation sequences. In Versions 9.0 through 12.1, such positional forms were included in the code chart for Mongolian, but have been removed from the code charts starting with Version 13.0, with the intent that they be shown instead in a publication dedicated to the details of the Mongolian text model.

24.1.13 Block Headers

The code charts are segmented by the format tooling into blocks. (See Definition D10b in *Section 3.4, Characters and Encoding*.) The page headers for the code charts are based on the normative values of the Block property defined in Blocks.txt in the Unicode Character Database, with a few exceptions. For example, the ASCII and Latin-1 ranges have their block headers adjusted editorially to reflect the presence of C0 and C1 control characters in those ranges. This means that the Block property value for the block associated with the range U+0080..U+00FF is "Latin-1 Supplement", but the block header used in the code charts is "C1 Controls and Latin-1 Supplement".

The start and end code points printed in the block headers in the code charts and character names list reflect the ranges that are printed *on that page*, and thus should not be confused with the normative ranges listed in Blocks.txt.

On occasion, the code chart format tooling also introduces artificial block headers to enable the display of code charts for noncharacters that are outside the range of any normative block range. For example, the two noncharacters U+3FFFE..U+3FFFF are artificially displayed in a code chart with a block header "Unassigned", showing a range U+3FF80..U+3FFFF.

As a result of these and other editorial considerations, implementers are cautioned not to attempt to pull block range values from the code charts, nor to attempt to parse them from the NamesList.txt file in the Unicode Character Database. Instead, normative values for block ranges and names should always depend on Blocks.txt.

24.1.14 Subheads

The character names list contains a number of informative subheads that help divide up the list into smaller sublists of similar characters. For example, in the Miscellaneous Symbols block, U+2600..U+26FF, there are subheads for "Astrological symbols," "Chess symbols," and so on. Such subheads are editorial and informative; they should not be taken as providing any definitive, normative status information about characters in the sublists they mark or about any constraints on what charac-

ters could be encoded in the future at reserved code points within their ranges. The subheads are subject to change.

24.2 CJK and Other Ideographs

The code charts for CJK and Tangut ideographs differ significantly from those for other characters in the standard.

24.2.1 CJK Unified Ideographs

Character names are not provided for any of the code charts of CJK Unified Ideograph character blocks, because the name of a unified ideograph simply consists of its Unicode code point preceded by CJK UNIFIED IDEOGRAPH-.

In other code charts, each character is shown with a single representative glyph, but in the code charts for CJK Unified and Compatibility Ideographs, each character may have multiple representative glyphs. Each character is shown with as many representative glyphs as there are Ideographic Research Group (IRG) sources defined for that character. The representative glyph for each IRG source is not necessarily the only preferred glyph for the corresponding region, and developers are therefore encouraged to refer to regional standards or typographical conventions to determine the appropriate glyph. Each representative glyph is accompanied with its source reference provided in alphanumeric form. Altogether, there are eleven IRG sources, as shown in *Table 24-1*. Data for these IRG sources are documented in Unicode Standard Annex #38, "Unicode Han Database (Unihan)."

Table 24-1. IRG Sources

Name	Source Identity
G source	China PRC and Singapore
H source	Hong Kong SAR
J source	Japan
KP source	North Korea
K source	South Korea
M source	Macao SAR
S source	SAT
T source	TCA
UK source	UK
U source	Unicode
V source	Vietnam

To assist in reference and lookup, each CJK Unified Ideograph is accompanied by a representative glyph of its Unicode radical and by its Unicode radical-stroke counts. These are printed directly underneath the Unicode code point for the character. A

radical-stroke index to all of the CJK ideographs is also provided separately on the Unicode website.

Chart for the Main CJK Block. The format for the CJK Unified Ideographs block (U+4E00..U+9FFF) is illustrated in *Figure 24-1*. The representative glyphs are arranged under the headers *C*, *J*, *K*, and *V*. Sources G, H, and T are grouped under the header *C*. Sources K and KP are grouped under the header *K*. The J and V sources are listed under their respective headers. Each row contains positions for all seven sources, and if a particular source is undefined for CJK Unified Ideographs, that position is left blank in the row. The gray vertical lines in *Figure 24-1* are used here to show how the sources are grouped under the *C*, *J*, *K*, and *V* headers.

Figure 24-1. CJK Chart Format for the Main CJK Block

If any of the M, U, UK or S sources are present, they are shown on a line by themselves below the G, H, T or J source position, respectively, as illustrated in *Figure 24-2*. Note that this block does not currently contain any characters with UK or S sources.

Figure 24-2. CJK Chart Format for M or U Source

If there are no other sources, the M, U, UK or S sources are shown in the G, H, T or J source position, respectively, as illustrated in *Figure 24-3*.

Figure 24-3. CJK Chart Format for Lone M or U Source

Charts for CJK Extensions. The code charts for all of the extension blocks for CJK Unified Ideographs use a more condensed format for character entries. That format dispenses with the *C*, *J*, *K*, and *V* headers and leaves no holes for undefined sources. For those blocks, sources are always shown in the following order: G, T, J, K, KP, V, H, M, U, UK, and S. The first letters of the source reference serve as a source tag.

The multicolumn code charts for CJK Extension A use the condensed format with three source columns per entry, and with entries arranged in three columns per page. An entry may have additional rows, if required, as illustrated in *Figure 24-4* for CJK Extension A.

Figure 24-4. CJK Chart Format for CJK Extension A

The multicolumn code charts for all of the other extension blocks for CJK Unified Ideographs currently use the condensed format with two source columns per entry, and with entries arranged in four columns per page. An entry may have additional rows if required.

The multicolumn code charts for the CJK Unified Ideographs Extension B block (U+20000..U+2A6DF) were introduced in Version 5.2 of the standard. From Version 6.1 through 13.0 of the standard, those multicolumn code charts had the additional idiosyncrasy that the first source shown always corresponded to the "UCS2003" representative glyph. Those representative glyphs were the only ones used up through Version 5.1 of the standard for that block, and have since been archived as a separate, archival code chart with a single representative glyph for each character.

24.2.2 Compatibility Ideographs

The format of the code charts for the CJK Compatibility Ideograph blocks is largely similar to the CJK chart format for Extension A, as illustrated in *Figure 24-5*. However, several additional notational elements described in *Section 24.1, Character Names List* are used. In particular, for each CJK compatibility ideograph other than the small list of unified ideographs included in these charts, a canonical decomposition is shown. The ideographic variation sequence for each compatibility CJK ideograph is listed below the canonical decomposition, introduced with a tilde sign.

Figure 24-5. CJK Chart Format for Compatibility Ideographs
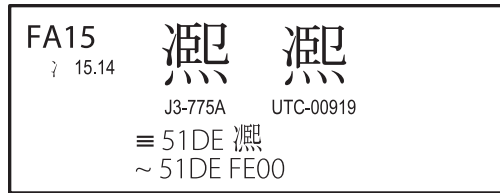
The twelve CJK unified ideographs in the CJK Compatibility Ideographs block have no canonical decompositions or corresponding ideographic variation sequences; instead, each is clearly labeled with an annotation identifying it as a CJK unified ideograph.

Figure 24-6. Annotations Identifying CJK Unified Ideographs
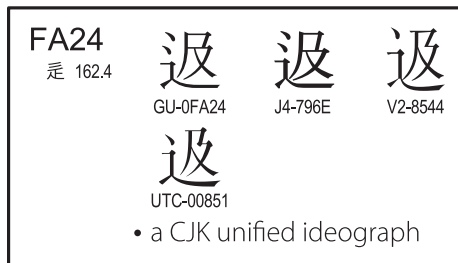
Character names are not provided for any CJK Compatibility Ideograph blocks because the name of a compatibility ideograph simply consists of its Unicode code point preceded by CJK COMPATIBILITY IDEOGRAPH-.

24.2.3 Tangut Ideographs

Code charts for Tangut ideographs use the same condensed format as the code charts for CJK Extension A, but with a single source column per entry, and with entries arranged in five columns per page.

Character names are not provided for any of the code charts of Tangut character blocks; the name of each Tangut ideograph simply consists of its Unicode code point preceded by TANGUT IDEOGRAPH-.

24.3 Hangul Syllables

As in the case of CJK Unified Ideographs, a character names list is not provided for the online chart of characters in the Hangul Syllables block, U+AC00..U+D7AF, because the name of a Hangul syllable can be determined by algorithm as described in *Section 3.12, Conjoining Jamo Behavior*. The short names used in that algorithm are listed in the code charts as aliases in the Hangul Jamo block, U+1100..U+11FF, as well as in Jamo.txt in the Unicode Character Database.

Appendix A

Notational Conventions

This appendix describes the typographic conventions, the extended BNF, and the conventions for describing rendering rules that are used throughout this core specification.

A.1 Typographic Conventions

A.1.1 Code Points

In running text, an individual Unicode code point is expressed as U+*n*, where *n* is four to six hexadecimal digits, using the digits 0–9 and uppercase letters A–F (for 10 through 15, respectively). Leading zeros are omitted, unless the code point would have fewer than four hexadecimal digits—for example, U+0001, U+0012, U+0123, U+1234, U+12345, U+102345.

- U+0416 is the Unicode code point for the character named CYRILLIC CAPITAL LETTER ZHE.

The *U+* may be omitted for brevity in tables or when denoting ranges. The *U+* is obligatorily omitted when this code point convention is used in rule NR2, for cases where characters have names algorithmically derived from their code points. See "Unicode Name Property" in *Section 4.8, Name*.

A range of Unicode code points is expressed as U+*xxxx*–U+*yyyy* or U+*xxxx*..U+*yyyy*, where *xxxx* and *yyyy* are the first and last Unicode values in the range, and the en dash or two dots indicate a contiguous range inclusive of the endpoints. For ranges involving supplementary characters, the code points in the ranges are expressed with five or six hexadecimal digits.

- The range U+0900–U+097F contains 128 Unicode code points.
- The Plane 16 private-use characters are in the range U+100000..U+10FFFD.

A.1.2 Character Names

In running text, a formal Unicode name is shown in small capitals (for example, GREEK SMALL LETTER MU), and alternative names (aliases) appear in italics (for example, *umlaut*). Italics are also used to refer to a text element that is not explicitly encoded (for example, *pasekh alef*) or to set off a non-English word (for example, the Welsh word *ynghyd*).

For more information on Unicode character names, see *Section 4.8, Name*.

For notational conventions used in the code charts, see *Section 24.1, Character Names List*.

A.1.3 Character Blocks

When referring to the normative names of character blocks in the text of the standard, the character block name is titlecased and is used with the term "block." For example:

the Latin Extended-B block

Optionally, an exact range for the character block may also be cited:

the Alphabetic Presentation Forms block (U+FB00..U+FB4F)

These references to normative character block names should not be confused with the headers used throughout the text of the standard, particularly in the block description chapters, to refer to particular ranges of characters. Such headers may be abbreviated in various ways and may refer to subranges within character blocks or ranges that cross character block boundaries. For example:

Latin Ligatures: U+FB00–U+FB06

The definitive list of normative character block names is Blocks.txt in the Unicode Character Database.

A.1.4 Sequences

A sequence of two or more code points may be represented by a comma-delimited list, set off by angle brackets. For this purpose, angle brackets consist of U+003C LESS-THAN SIGN and U+003E GREATER-THAN SIGN. Spaces are optional after the comma, and U+ notation for the code point is also optional—for example, "<U+0061, U+0300>".

When the usage is clear from the context, a sequence of characters may be represented with generic short names, as in "<a, grave>", or the angle brackets may be omitted.

In contrast to sequences of code points, a sequence of one or more code *units* may be represented by a list set off by angle brackets, but without comma delimitation or U+ notation. For example, the notation "<nn nn nn nn>" represents a sequence of bytes, as for the UTF-8 encoding form of a Unicode character. The notation "<nnnn nnnn>" represents a sequence of 16-bit code units, as for the UTF-16 encoding form of a Unicode character.

A.1.5 Properties and Property Values

The names of properties and property values appear in titlecase, with words connected by an underscore—for example, General_Category or Uppercase_Letter. In some instances, short names are used, such as gc = Lu, which is equivalent to General_Category = Uppercase_Letter. Long and short names for all properties and property values are defined in the Unicode Character Database; see also *Section 3.5, Properties*.

Occasionally, and especially when discussing character properties that have single words as names, such as *age* and *block,* the names appear in lowercase italics.

A.1.6 Miscellaneous

Phonemic transcriptions are shown between slashes, as in Khmer /khnyom/.

Phonetic transcriptions are shown between square brackets, using the International Phonetic Alphabet. (Full details on the IPA can be found on the International Phonetic Association's website, https://www.internationalphoneticassociation.org/.)

A leading asterisk is used to represent an incorrect or nonoccurring linguistic form.

In this specification, the word "Unicode" when used alone as a noun refers to the Unicode Standard.

Unambiguous dates of the current common era, such as 1999, are unlabeled. In cases of ambiguity, CE is used. Dates before the common era are labeled with BCE.

The term *byte*, as used in this standard, always refers to a unit of eight bits. This corresponds to the use of the term *octet* in some other standards.

A.1.7 Operators

Operators used in this standard are listed in *Table A-1*.

Table A-1. Operators

Symbol	Meaning
→	is transformed to, or behaves like
↛	is not transformed to
¬	logical not

A.2 Extended BNF

The Unicode Standard and technical reports use an extended BNF format for describing syntax. This format uses elements from the regular expression syntax specified in Unicode Technical Standard #18, "Unicode Regular Expressions"; however, a BNF is not a regular expression, and may be interpreted differently even when looking like one. As different conventions are used for BNF, *Table A-2* lists the notation used here.

Table A-2. Extended BNF

Symbols	Meaning
x := ...	production rule
x y	the sequence consisting of x then y
x*	zero or more occurrences of x
x?	zero or one occurrence of x
x+	one or more occurrences of x
x | y	either x or y
(x)	for grouping
{ x }	equivalent to (x)?
"abc"	string literals ("_" is sometimes used to denote space for clarity)
'abc'	string literals (alternative form)
sot	start of text
eot	end of text
\u{HHHHHH}	Unicode code points within string literals or character classes. Between one and six hexadecimal digits; maximum \u{10FFFF}.
\uHHHH	Unicode BMP code points within string literals or character classes. Exactly four hexadecimal digits.
U+HHHHHH	Unicode code point literal: equivalent to "\u{HHHHHH}". Between four and six hexadecimal digits; maximum U+10FFFF.
U-00HHHHHH	Unicode code point literal: equivalent to "\u{HHHHHH}". Exactly six hexadecimal digits after the initial two zeroes; maximum U+10FFFF. This format was used in ISO 10646 but is now obsolete.
H	Hexadecimal digit, 0-9 or A-F
[...], \p{...}	code point or character class (syntax below)

In other environments, such as programming languages or markup, alternative notation for sequences of code points or code units may be used.

A.2.1 Character Classes

A *code point class* is a set of code points. When the code points are all assigned characters, it can also be referred to as a *character class*. Its specification can be based on any of the following:

- A literal code point or a range of literal code points.
- A set of code points having a given value for a given Unicode character property, as defined in the Unicode Character Database (see PropertyAliases.txt and PropertyValueAliases.txt).
- Set operations on character classes.

Further extensions to this specification of character classes are used in some Unicode Standard Annexes and Unicode Technical Reports. Such extensions are described in those documents, as appropriate.

A partial formal BNF syntax for character classes as used in this standard is given by the following:

```
CHARACTER_CLASS := '[' COMPLEMENT? SET ']' | '\p{' PROP_SPEC
'}'

COMPLEMENT := '^'

SET := ITEM (SET_EXTEND)*

ITEM := LITERAL (RANGE_OPERATOR LITERAL)? | CHARACTER_CLASS

RANGE_OPERATOR := '-' | '..'

SET_EXTEND := SET_OPERATOR CHARACTER_CLASS | ','? ITEM

SET_OPERATOR := '--'

PROP_SPEC := PROP_NAME (RELATION PROP_VALUE)?

RELATION := '=' | '≠'
```

If COMPLEMENT is specified, the resulting code point set is the set of all Unicode code points (U+0000..U+10FFFF) except the code points given by SET. A LITERAL can be a Unicode code point escape sequence, a Unicode code point literal, or a character itself. The operator "--" indicates set difference (older documents may use "-"). A PROP_NAME must be a valid Unicode property name or alias. A PROP_VALUE must be a valid property value for the PROP_NAME it is used with. If a PROP_NAME is used by itself, without a RELATION and PROP_VALUE, the property must be a Boolean property, the relation is assumed to be "=" and the value to be True.

In prose where the context makes clear that a property-based character class is being discussed, `\p{PROP_NAME=PROP_VALUE}` may be simplified to `PROP_NAME=PROP_VALUE`.

Whenever any character could be interpreted as a syntax character, it must be escaped. If a space character is used as a literal, it is escaped. The interpretation of spaces differs from that in regular expressions, so that in the examples below spaces have to be removed in order to obtain equivalent regular expressions. Examples are found in *Table A-3*.

<div align="center">

Table A-3. Character Class Examples

</div>

Syntax	Matches
`[a-z]`	English lowercase letters
`[a-z -- c]`	English lowercase letters except for c
`[0-9]`	European decimal digits
`[\u0030-\u0039]`	(same as above, using Unicode escapes)
`[0-9 A-F a-f]`	hexadecimal digits
`[\p{gc=Letter} \p{gc=Nonspacing_Mark}]`	all letters and nonspacing marks
`[\p{gc=L} \p{gc=Mn}]`	(same as above, using abbreviated notation)
`[^\p{gc=Unassigned}]`	all assigned Unicode characters
`[\u{A980}-\u{A9DF} -- \p{gc=Unassigned}]`	all assigned characters in the main Javanese range
`[\p{Alphabetic}]`	all alphabetic characters
`[^\p{Line_Break=Infix_Numeric}]`	all code points that do not have the line break property of Infix_Numeric

For more information about character classes, see Unicode Technical Standard #18, "Unicode Regular Expressions."

A.3 Rendering

A figure such as *Figure A-1* depicts how a sequence of characters is typically rendered.

Figure A-1. Example of Rendering

$$A + \ddot{\circ} \rightarrow \ddot{A}$$
$$\scriptsize 0041 \quad\quad 0308$$

The sequence under discussion is depicted on the left of the arrow, using representative glyphs and code points below them. A possible rendering of that sequence is depicted on the right side of the arrow.

Appendix B

Unicode Publications and Resources

This appendix provides information about the Unicode Consortium and its activities, particularly regarding publications other than the Unicode Standard. The Unicode Consortium publishes a number of technical standards and technical reports. *Appendix B.2, Unicode Publications* describes the kinds of reports in more detail.

The Unicode website also has many useful online resources. *Appendix B.3, Other Unicode Online Resources*, provides a guide to the kinds of information available.

B.1 The Unicode Consortium

The Unicode Consortium was incorporated in January 1991, under the name Unicode, Inc., to promote the Unicode Standard as an international encoding system for information interchange, to aid in its implementation, and to maintain quality control over future revisions.

To further these goals, the Unicode Consortium cooperates with the Joint Technical Committee 1 of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1). It holds a Class C liaison membership with ISO/IEC JTC1/SC2; it participates in the work of both JTC1/SC2/WG2 (the technical working group for the subcommittee within JTC1 responsible for character set encoding) and the Ideographic Research Group (IRG) of WG2. The Consortium is a member company of the InterNational Committee for Information Technology Standards, Technical Committee L2 (INCITS/L2), an accredited U.S. standards organization. Many members of the Unicode Consortium have representatives in many countries who also work with other national standards bodies. In addition, a number of organizations are Liaison Members of the Consortium. For a list, see the Unicode website.

Membership in the Unicode Consortium is open to organizations and individuals anywhere in the world who support the Unicode Standard and who would like to assist in its extension and widespread implementation. Full, Institutional, Supporting, and Associate Members represent a broad spectrum of corporations and organizations in the computer and information processing industry. For a list, see the Unicode website. The Consortium is supported financially solely through membership dues and donations.

B.1.1 The Unicode Technical Committee

The Unicode Technical Committee (UTC) is the working group within the Consortium responsible for the creation, maintenance, and quality of the Unicode Standard. The UTC follows an open process in developing the Unicode Standard and its other technical publications. It coordinates and reviews all technical input to these documents and decides their contents. For more information on the UTC and the process by which the Unicode Standard and the other technical publications are developed, see:

   https://www.unicode.org/consortium/utc.html

B.1.2 Other Activities

Going beyond developing technical standards, the Unicode Consortium acts as registration authority for the registration of script identifiers under ISO 15924, and it has a

technical committee dedicated to the maintenance of the Unicode Common Locale Data Repository (CLDR). The repository contains a large and rapidly growing body of data used in the locale definition for software internationalization. For further information about these and other activities of the Unicode Consortium, visit:

https://www.unicode.org

B.2 Unicode Publications

In addition to the Unicode Standard, the Unicode Consortium publishes Unicode Technical Standards and Unicode Technical Reports. These materials are published as electronic documents only and, unlike Unicode Standard Annexes, do not form part of the Unicode Standard.

A *Unicode Standard Annex* (UAX) forms an integral part of the Unicode Standard, but is published online as a separate document. The Unicode Standard may require conformance to normative content in a Unicode Standard Annex, if so specified in the Conformance chapter of that version of the Unicode Standard. The version number of a UAX document is always the same as the version of the Unicode Standard of which it forms a part.

A *Unicode Technical Standard* (UTS) is an independent specification. Conformance to the Unicode Standard does not imply conformance to any UTS.

A *Unicode Technical Report* (UTR) contains informative material. Conformance to the Unicode Standard does not imply conformance to any UTR. Other specifications, however, are free to make normative references to a UTR.

In the past, some normative material was published as Unicode Technical Reports. Currently, however, such material is published either as a Unicode Technical Standard or a Unicode Standard Annex.

The Unicode website is the source for the most current version of all three categories of technical reports:

  https://www.unicode.org/reports/

B.3 Other Unicode Online Resources

The Unicode Consortium provides a number of online resources for obtaining information and data about the Unicode Standard as well as updates and corrigenda.

B.3.1 Unicode Online Resources

Unicode Website.

https://www.unicode.org

Charts. The charts section of the website provides code charts for all of the Unicode characters, plus specialized charts for normalization, collation, case mapping, script names, and CJK unified ideographs.

https://www.unicode.org/charts/

Character Index. Online index by character name, to look up Unicode code points. This index also makes it easy to look up the location of scripts in the standard, and indexes common alternative names for characters as well.

https://www.unicode.org/charts/charindex.html

Character Statistics. Access to tables containing raw character counts for every version of the Unicode Standard. This page also provides access to counts of emoji for various versions, and to the history of addition of scripts to the standard.

https://www.unicode.org/versions/stats/

Conferences. The Internationalization and Unicode Conferences are of particular value to anyone implementing the Unicode Standard or working on internationalization. A variety of tutorials and conference sessions cover current topics related to the Unicode Standard, the World Wide Web, software, internationalization, and localization.

https://www.unicode.org/conference/

E-mail Discussion List. Subscription instructions for the public e-mail discussion list are posted on the Unicode website.

Emoji. This page contains a collection of information about emoji resources and the Unicode Emoji Subcommittee.

https://www.unicode.org/emoji/techindex.html

Emoji Charts. This section of the Unicode website collects together a number of charts that illustrate various features of the emoji characters in the standard. Some of these charts can be considered as adjuncts to the regular code charts. They show emoji with colorful glyphs, as they are displayed on many platforms.

https://www.unicode.org/emoji/charts/

FAQ (Frequently Asked Questions). The FAQ pages provide an invaluable resource for understanding the Unicode Standard and its implications for users and implementers.

https://www.unicode.org/faq/

Glossary. Online listing of definitions for technical terms used in the Unicode Standard and other publications of the Unicode Consortium.

https://www.unicode.org/glossary/

Online Unicode Character Database. This page supplies information about the online Unicode Character Database (UCD), including links to documentation files and the most up-to-date version of the data files, as well as instructions on how to access any particular version of the UCD.

https://www.unicode.org/ucd/

Online Unihan Database. The online Unihan Database provides interactive access to all of the property information associated with CJK ideographs in the Unicode Standard.

https://www.unicode.org/charts/unihan.html

Pipeline. This page lists characters, standardized variation sequences, and named character sequences which have reached some level of approval and/or are in international balloting, but which have not yet been published in a version of the Unicode Standard. The pipeline provides some visibility about what characters will soon be in the standard.

https://www.unicode.org/alloc/Pipeline.html

Policies. These pages describe Unicode Consortium policies on technical stability. The stability policies are particularly important for implementers, documenting invariants for the Unicode Standard that allow implementations to be compatible with future and past versions.

https://www.unicode.org/policies/stability.html

References. This online page lists sources and up-to-date references for the Unicode Standard, as well as resources by script.

https://www.unicode.org/references/

Roadmap. This section of the Unicode website provides a roadmap for planning future allocation of scripts and major blocks of symbols. The roadmap is organized by plane, and provides information about the locations of published, approved, and proposed blocks, often with links to current proposals. The roadmap provides the long term perspective on future work by the encoding committees.

https://www.unicode.org/roadmaps/

Unicode Common Locale Data Repository (CLDR). Machine-readable repository, in XML format, of locale information for use in application and system development.

https://cldr.unicode.org/

Updates and Errata. This page lists periodic updates with corrections of typographic errors and new clarifications of the text.

https://www.unicode.org/errata/

Versions. This page describes the version numbering used in the Unicode Standard, the nature of the Unicode character repertoire, and ways to cite and reference the Unicode Standard, the Unicode Character Database, and Unicode Technical Reports. It also specifies the exact contents of each and every version of the Unicode Standard, back to Unicode 1.0.0.

https://www.unicode.org/versions/

Where Is My Character?. This page provides basic guidance to finding Unicode characters, especially those whose glyphs do not appear in the charts, or that are represented by sequences of Unicode characters.

https://www.unicode.org/standard/where/

B.3.2 How to Contact the Unicode Consortium

The best way to contact the Unicode Consortium to obtain membership information is via the website:

https://home.unicode.org/membership/membership-levels/

To contact the Unicode Consortium regarding other issues:

https://home.unicode.org/connect/contact-unicode/

The website also lists the current postal address and telephone number.

Appendix C

Relationship to ISO/IEC 10646

The Unicode Consortium maintains a strong working relationship with ISO/IEC JTC1/SC2/WG2, the working group developing International Standard 10646. Today both organizations are firmly committed to maintaining the synchronization between the Unicode Standard and ISO/IEC 10646. Each standard nevertheless uses its own form of reference and, to some degree, separate terminology. This appendix gives a brief history and explains how the standards are related.

C.1 History

Having recognized the benefits of developing a single universal character code standard, members of the Unicode Consortium worked with representatives from the International Organization for Standardization (ISO) during the summer and fall of 1991 to pursue this goal. Meetings between the two bodies resulted in mutually acceptable changes to both Unicode Version 1.0 and the first ISO/IEC Draft International Standard DIS 10646.1, which merged their combined repertoire into a single numerical character encoding. This work culminated in *The Unicode Standard, Version 1.1*.

ISO/IEC 10646-1:1993, *Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane*,was published in May 1993 after final editorial changes were made to accommodate the comments of voting members. *The Unicode Standard, Version 1.1*, reflected the additional characters introduced from the DIS 10646.1 repertoire and incorporated minor editorial changes.

Merging *The Unicode Standard, Version 1.0,*and DIS 10646.1 consisted of aligning the numerical values of identical characters and then filling in some groups of characters that were present in DIS 10646.1, but not in the Unicode Standard. As a result, the encoded characters (code points and names) of ISO/IEC 10646-1:1993 and *The Unicode Standard, Version 1.1*, are precisely the same.

Versions 2.0, 2.1, and 3.0 of the Unicode Standard successively added more characters, matching a series of amendments to ISO/IEC 10646-1. *The Unicode Standard, Version 3.0*, is precisely aligned with the second edition of ISO/IEC 10646-1, known as ISO/IEC 10646-1:2000.

In 2001, Part 2 of ISO/IEC 10646 was published as ISO/IEC 10646-2:2001. Version 3.1 of the Unicode Standard was synchronized with that publication, which added supplementary characters for the first time. Subsequently, Versions 3.2 and 4.0 of the Unicode Standard added characters matching further amendments to both parts of ISO/IEC 10646. *The Unicode Standard, Version 4.0*, is precisely aligned with the third version of ISO/IEC 10646 (first edition), published as a single standard merging the former two parts: ISO/IEC 10646:2003.

Versions 4.1 and 5.0 of the Unicode Standard added characters matching Amendments 1 and 2 to ISO/IEC 10646:2003. Version 5.0 also added four characters for Sindhi support from Amendment 3 to ISO/IEC 10646:2003. Version 5.1 added the rest of the characters from Amendment 3 and all of the characters from Amendment 4 to ISO/IEC 10646:2003. Version 5.2 added all of the characters from Amendments 5 and 6 to ISO/IEC 10646:2003. Version 6.0 added all of the characters from Amendments 7 and 8 to ISO/IEC 10646:2003.

In 2010, ISO approved republication of ISO/IEC 10646 as a second edition, ISO/IEC 10646:2011, consolidating all of the contents of Amendments 1 through 8 to the 2003 first edition. *The Unicode Standard, Version 6.0* is aligned with that second edition of the International Standard, with the addition of U+20B9 INDIAN RUPEE SIGN, accelerated into Version 6.0 based on approval for the third edition of ISO/IEC 10646.

The Unicode Standard, Version 6.1 is aligned with the third edition of the International Standard: ISO/IEC 10646:2012. The third edition was approved for publication without an intervening amendment to the second edition. *The Unicode Standard, Version 6.2* added a single character, U+20BA TURKISH LIRA SIGN. Version 6.3 added five more characters, including new bidirectional format controls.

The Unicode Standard, Version 7.0 is aligned with Amendments 1 and 2 to ISO/IEC 10646:2012. Those amendments include the six characters which were added in Version 6.2 and Version 6.3, as well as many others. Version 7.0 also includes U+20BD RUBLE SIGN, accelerated into Version 7.0 based on approval for the fourth edition of ISO/IEC 10646.

The Unicode Standard, Version 8.0 is aligned with Amendment 1 of ISO/IEC 10646:2014, the fourth edition of ISO/IEC 10646. Version 8.0 also includes U+20BE LARI SIGN, nine additional CJK unified ideographs, and 41 emoji characters, based on approval for Amendment 2 to the fourth edition of ISO/IEC 10646.

The Unicode Standard, Version 9.0 is aligned with Amendments 1 and 2 to ISO/IEC 10646:2014, the fourth edition of ISO/IEC 10646. Version 9.0 also includes the Adlam script (87 characters), the Newa script (92 characters), 72 additional emoji characters, 19 television symbols, two other pictographic symbols, and one other punctuation mark, based on approval for the fifth edition of ISO/IEC 10646.

The Unicode Standard, Version 10.0 is aligned with ISO/IEC 10646:2017, the fifth edition of ISO/IEC 10646. Version 10.0 also includes three other Zanabazar Square characters, 285 hentaigana, and 56 emoji characters, based on approval for Amendment 1 to the fifth edition of ISO/IEC 10646.

The Unicode Standard, Version 11.0 is aligned with Amendment 1 to ISO/IEC 10646:2017, the fifth edition of ISO/IEC 10646. Version 11.0 also includes 46 Mtavruli Georgian capital letters, 5 urgently needed CJK unified ideographs, and 66 emoji characters, based on approval for Amendment 2 to the fifth edition of ISO/IEC 10646.

The Unicode Standard, Version 12.0 is aligned with Amendments 1 and 2 to ISO/IEC 10646:2017, the fifth edition of ISO/IEC 10646. Version 12.0 also includes U+1E94B ADLAM NASALIZATION MARK and 61 emoji characters, based on approval for the sixth edition of ISO/IEC 10646.

The Unicode Standard, Version 13.0 is aligned with ISO/IEC 10646:2020, the sixth edition of ISO/IEC 10646.

The Unicode Standard, Version 15.0 is aligned with Amendment 1 to ISO/IEC 10646:2020, the sixth edition of ISO/IEC 10646.

The Unicode Standard, Version 16.0 is aligned with Amendment 2 to ISO/IEC 10646:2020, the sixth edition of ISO/IEC 10646.

Table C-1 gives the timeline for these efforts.

Table C-1. Timeline

Year	Version	Summary
1989	DP 10646	Draft proposal, independent of Unicode
1990	Unicode Prepublication	Prepublication review draft
1990	DIS-1 10646	First draft, independent of Unicode
1991	Unicode 1.0	Edition published by Addison-Wesley
1992	Unicode 1.0.1	Modified for merger compatibility
1992	DIS-2 10646	Second draft, merged with Unicode
1993	IS 10646-1:1993	Merged standard
1993	Unicode 1.1	Revised to match IS 10646-1:1993
1995	10646 amendments	Korean realigned, plus additions
1996	Unicode 2.0	Synchronized with 10646 amendments
1998	Unicode 2.1	Added euro sign and corrigenda
1999	10646 amendments	Additions
2000	Unicode 3.0	Synchronized with 10646 second edition
2000	IS 10646-1:2000	10646 part 1, second edition, publication with amendments to date
2001	IS 10646-2:2001	10646 part 2 (supplementary planes)
2001	Unicode 3.1	Synchronized with 10646 part 2
2002	Unicode 3.2	Synchronized with Amd 1 to 10646 part 1
2003	Unicode 4.0	Synchronized with 10646 third version
2003	IS 10646:2003	10646 third version (first edition), merging the two parts
2005	Unicode 4.1	Synchronized with Amd 1 to 10646:2003
2006	Unicode 5.0	Synchronized with Amd 2 to 10646:2003, plus Sindhi additions
2008	Unicode 5.1	Synchronized with Amd 3 and Amd 4 to 10646:2003
2009	Unicode 5.2	Synchronized with Amd 5 and Amd 6 to 10646:2003
2010	Unicode 6.0	Synchronized with 10646 second edition of third version, plus the Indian rupee sign
2011	IS 10646:2011	10646 second edition of third version
2012	Unicode 6.1	Synchronized with 10646 third edition of third version
2012	IS 10646:2012	10646 third edition of third version
2012	Unicode 6.2	Added Turkish lira sign

2013	Unicode 6.3	Added several bidirectional controls
2014	Unicode 7.0	Synchronized with Amd 1 and Amd 2 to 10646:2012, plus the ruble sign
2014	IS 10646:2014	10646 fourth edition to third version
2015	Unicode 8.0	Synchronized with Amd 1 to 10646:2014, plus 51 additional characters
2016	Unicode 9.0	Synchronized with Amd 1 and Amd 2 to 10646:2014, plus 273 additional characters
2017	Unicode 10.0	Synchronized with 10646 fifth edition of third version, plus 344 additional characters
2017	IS 10646:2017	10646 fifth edition to third version
2018	Unicode 11.0	Synchronized with Amd 1 to 10646:2017, plus 117 additional characters
2019	Unicode 12.0	Synchronized with Amd 1 and Amd 2 to 10646:2017, plus 62 additional characters
2019	Unicode 12.1	Added Japanese era name reiwa
2020	Unicode 13.0	Synchronized with 10646 sixth edition of third version
2020	IS 10646:2020	10646 sixth edition to third version
2021	Unicode 14.0	Added 838 characters
2022	Unicode 15.0	Synchronized with Amd 1 to 10646:2020
2023	Unicode 15.1	Added 627 characters
2024	Unicode 16.0	Synchronized with Amd 2 to 10646:2020

C.1.1 Unicode 1.0

The combined repertoire presented in ISO/IEC 10646 is a superset of *The Unicode Standard, Version 1.0,* repertoire as amended by *The Unicode Standard, Version 1.0.1. The Unicode Standard, Version 1.0*, was amended by the *Unicode 1.0.1 Addendum* to make the Unicode Standard a proper subset of ISO/IEC 10646. This effort entailed both moving and eliminating a small number of characters.

C.1.2 Unicode 2.0

The Unicode Standard, Version 2.0, covered the repertoire of *The Unicode Standard, Version 1.1* (and IS 10646), plus the first seven amendments to IS 10646, as follows:

Amd. 1: UTF-16

Amd. 2: UTF-8

Amd. 3: Coding of C1 Controls

Amd. 4: Removal of Annex G: UTF-1

Amd. 5: Korean Hangul Character Collection

Amd. 6: Tibetan Character Collection

Amd. 7: 33 Additional Characters (Hebrew, Long S, Dong)

In addition, *The Unicode Standard, Version 2.0*, covered Technical Corrigendum No. 1 (on renaming of AE LIGATURE to LETTER) and such Editorial Corrigenda to ISO/IEC 10646 as were applicable to the Unicode Standard. The euro sign and the object replacement character were added in Version 2.1, per amendment 18 of ISO/IEC 10646-1.

C.1.3 Unicode 3.0

The Unicode Standard, Version 3.0, is synchronized with the second edition of ISO/ IEC 10646-1. The latter contains all of the published amendments to 10646-1; the list includes the first seven amendments, plus the following:

Amd. 8: Addition of Annex T: Procedure for the Unification and Arrangement of CJK Ideographs

Amd. 9: Identifiers for Characters

Amd. 10: Ethiopic Character Collection

Amd. 11: Unified Canadian Aboriginal Syllabics Character Collection

Amd. 12: Cherokee Character Collection

Amd. 13: CJK Unified Ideographs with Supplementary Sources (Horizontal Extension)

Amd. 14: Yi Syllables and Yi Radicals Character Collection

Amd. 15: Kangxi Radicals, Hangzhou Numerals Character Collection

Amd. 16: Braille Patterns Character Collection

Amd. 17: CJK Unified Ideographs Extension A (Vertical Extension)

Amd. 18: Miscellaneous Letters and Symbols Character Collection (which includes the euro sign)

Amd. 19: Runic Character Collection

Amd. 20: Ogham Character Collection

Amd. 21: Sinhala Character Collection

Amd. 22: Keyboard Symbols Character Collection

Amd. 23: Bopomofo Extensions and Other Character Collection

Amd. 24: Thaana Character Collection

Amd. 25: Khmer Character Collection

Amd. 26: Myanmar Character Collection

Amd. 27: Syriac Character Collection

Amd. 28: Ideographic Description Characters

Amd. 29: Mongolian

Amd. 30: Additional Latin and Other Characters

Amd. 31: Tibetan Extension

The second edition of ISO/IEC 10646-1 also contains the contents of Technical Corrigendum No. 2 and all the Editorial Corrigenda to the year 2000.

C.1.4 Unicode 4.0

The Unicode Standard, Version 4.0, is synchronized with the third version of ISO/IEC 10646. The third version of ISO/IEC 10646 is the result of the merger of the second edition of Part 1 (ISO/IEC 10646-1:2000) with the first edition of Part 2 (ISO/IEC 10646-2:2001) into a single publication. The third version incorporates the published amendments to 10646-1 and 10646-2:

Amd. 1 (to part 1): Mathematical symbols and other characters

Amd. 2 (to part 1): Limbu, Tai Le, Yijing, and other characters

Amd. 1 (to part 2): Aegean, Ugaritic, and other characters

The third version of ISO/IEC 10646 also contains all the Editorial Corrigenda to date.

C.1.5 Unicode 5.0

The Unicode Standard, Version 5.0, is synchronized with ISO/IEC 10646:2003 plus its first two published amendments:

Amd. 1: Glagolitic, Coptic, Georgian and other characters

Amd. 2: N'Ko, Phags-Pa, Phoenician and Cuneiform

Four Devanagari characters for the support of the Sindhi language (U+097B, U+097C, U+097E, U+097F) were added in Version 5.0 per Amendment 3 of ISO/IEC 10646.

C.1.6 Unicode 6.0

The Unicode Standard, Version 6.0, is synchronized with the second edition of ISO/IEC 10646. The second edition of the third version of ISO/IEC 10646 consolidates all of the repertoire additions from the published eight amendments of ISO/IEC 10646:2003. These include the first two amendments listed under Unicode 5.0, plus the following:

Amd. 3: Lepcha, Ol Chiki, Saurashtra, Vai, and other characters

Amd. 4: Cham, Game Tiles, and other characters

Amd. 5: Tai Tham, Tai Viet, Avestan, Egyptian Hieroglyphs, CJK Unified Ideographs Extension C, and other characters

Amd. 6: Javanese, Lisu, Meetei Mayek, Samaritan, and other characters

Amd. 7: Mandaic, Batak, Brahmi, and other characters

Amd. 8: Additional symbols, Bamum supplement, CJK Unified Ideographs Extension D, and other characters

One additional character, for the support of the new Indian currency symbol (U+20B9 INDIAN RUPEE SIGN), was accelerated into Version 6.0, based on its approval for the third edition of ISO/IEC 10646.

C.1.7 Unicode 7.0

The Unicode Standard, Version 7.0, is synchronized with the third edition of ISO/IEC 10646 plus its two published amendments:

Amd. 1: Linear A, Palmyrene, Manichaean, Khojki, Khudawadi, Bassa Vah, Duployan, and other characters

Amd. 2: Caucasian Albanian, Psalter Pahlavi, Mahajani, Grantha, Modi, Pahawh Hmong, Mende Kikakui, and other characters

One additional character, for the support of the new Russian currency symbol (U+20BD RUBLE SIGN), was accelerated into Version 7.0, based on its approval for the fourth edition of ISO/IEC 10646.

C.1.8 Unicode 8.0

The Unicode Standard, Version 8.0, is synchronized with the fourth edition of ISO/IEC 10646, plus its first published amendment:

Amd. 1: Cherokee supplement and other characters

An additional 51 characters were accelerated into Version 8.0, based on their approval for Amendment 2 to the fourth edition of ISO/IEC 10646. These include U+20BE LARI SIGN, for the support of the new Georgian currency symbol, nine additional CJK unified ideographs, and 41 emoji characters.

C.1.9 Unicode 9.0

The Unicode Standard, Version 9.0, is synchronized with the fourth edition of ISO/IEC 10646, plus its two published amendments:

Amd. 1: Cherokee supplement and other characters

Amd. 2: Bhaiksuki, Marchen, Tangut and other characters

An additional 273 characters were accelerated into Version 9.0, based on their approval for the fifth edition of ISO/IEC 10646. These include characters for the Adlam script and the Newa script, 72 emoji characters, 19 television symbols, and one other punctuation mark.

C.1.10 Unicode 10.0

The Unicode Standard, Version 10.0, is synchronized with the fifth edition of ISO/IEC 10646.

An additional 344 characters were accelerated into Version 10.0, based on their approval for Amendment 1 to the fifth edition of ISO/IEC 10646. These include three additional characters for the Zanabazar Square script, 285 hentaigana characters, and 56 emoji characters.

C.1.11 Unicode 11.0

The Unicode Standard, Version 11.0, is synchronized with the fifth edition of ISO/IEC 10646, plus its first published amendment:

Amd. 1: Dogra, Gunjala Gondi, Makasar, Medefaidrin, Indic Siyaq Numbers, and other characters

An additional 117 characters were accelerated into Version 11.0, based on their approval for Amendment 2 to the fifth edition of ISO/IEC 10646. These include

Mtavruli uppercase Georgian letters, five additional CJK unified ideographs, and 66 emoji characters.

C.1.12 Unicode 12.0

The Unicode Standard, Version 12.0, is synchronized with the fifth edition of ISO/IEC 10646, plus its two published amendments:

Amd. 1: Dogra, Gunjala Gondi, Makasar, Medefaidrin, Indic Siyaq Numbers, and other characters

Amd. 2: Nandinagari, Georgian extension, and other characters

An additional 62 characters were accelerated into Version 12.0, based on their approval for the sixth edition of ISO/IEC 10646. These include U+1E94B ADLAM NASALIZATION MARK and 61 emoji characters.

C.1.13 Unicode 13.0

The Unicode Standard, Version 13.0 is synchronized with the sixth edition of ISO/IEC 10646.

C.1.14 Unicode 14.0

The Unicode Standard, Version 14.0 adds 838 new characters, including nine unified CJK ideographs.

C.1.15 Unicode 15.0

The Unicode Standard, Version 15.0 adds 4,489 new characters, including 4,193 unified CJK ideographs. Version 15.1 adds 627 further characters, including 622 urgently needed unified CJK ideographs.

C.1.16 Unicode 16.0

The Unicode Standard, Version 16.0 adds 5,185 new characters. Version 16.0 is the first version for which the core specification has been published in interactive HTML. The HTML version is authoritative. For archival purposes, a PDF version is produced directly from the HTML.

The synchronization of *The Unicode Standard, Version 16.0*, with ISO/IEC 10646:2020 plus Amd 1 and Amd 2 means that the repertoire, encoding, and names of all characters are identical between the two standards at those version levels. All other changes to the text of 10646 that have a bearing on the Unicode Standard have been taken into account in the revision of the Unicode Standard.

C.2 Encoding Forms in ISO/IEC 10646

ISO/IEC 10646:2011 has significantly revised its discussion of encoding forms, compared to earlier editions of that standard. The terminology for encoding forms (and encoding schemes) in 10646 now matches exactly the terminology used in the Unicode Standard. Furthermore, 10646 is now described in terms of a codespace U+0000..U+10FFFF, instead of a 31-bit codespace, as in earlier editions. This convergence in codespace description has eliminated any discrepancies in possible interpretation of the numeric values greater than 0x10FFFF. As a result, this section now merely notes a few items of mostly historic interest regarding encoding forms and terminology.

UCS-4. UCS-4 stands for "Universal Character Set coded in 4 octets." It is now treated simply as a synonym for UTF-32, and is considered the canonical form for representation of characters in 10646.

UCS-2. UCS-2 stands for "Universal Character Set coded in 2 octets" and is also known as "the two-octet BMP form." It was documented in earlier editions of 10646 as the two-octet (16-bit) encoding consisting only of code positions for plane zero, the *Basic Multilingual Plane*. This documentation has been removed from ISO/IEC 10646:2011 and subsequent editions, and the term UCS-2 should now be considered obsolete. It no longer refers to an encoding form in either 10646 or the Unicode Standard.

C.2.1 Zero Extending

The character "A", U+0041 LATIN CAPITAL LETTER A, has the unchanging numerical value 41 hexadecimal. This value may be extended by any quantity of leading zeros to serve in the context of the following encoding standards and transformation formats (see *Table C-2*).

Table C-2. Zero Extending

Bits	Standard	Binary	Hex	Dec	Char
7	ASCII	1000001	41	65	A
8	8859-1	01000001	41	65	A
16	UTF-16	00000000 01000001	41	65	A
32	UTF-32, UCS-4	00000000 00000000 00000000 01000001	41	65	A

This design eliminates the problem of disparate values in all systems that use either of the standards and their transformation formats.

C.3 UTF-8 and UTF-16

C.3.1 UTF-8

The ISO/IEC 10646 definition of UTF-8 is identical to UTF-8 as described under Definition D92 in *Section 3.9, Unicode Encoding Forms*.

UTF-8 can be used to transmit text data through communications systems that assume that individual octets in the range of x00 to x7F have a definition according to ISO/IEC 4873, including a C0 set of control functions according to the 8-bit structure of ISO/IEC 2022. UTF-8 also avoids the use of octet values in this range that have special significance during the parsing of file name character strings in widely used file-handling systems.

C.3.2 UTF-16

The ISO/IEC 10646 definition of UTF-16 is identical to UTF-16 as described under Definition D91 in *Section 3.9, Unicode Encoding Forms*.

C.4 Synchronization of the Standards

Programmers and system users should treat the encoded character values from the Unicode Standard and ISO/IEC 10646 as identities, especially in the transmission of raw character data across system boundaries. The Unicode Consortium and ISO/IEC JTC1/SC2/WG2 are committed to maintaining the synchronization between the two standards.

However, the Unicode Standard and ISO/IEC 10646 differ in the precise terms of their conformance specifications. Any Unicode implementation will conform to ISO/IEC 10646, but because the Unicode Standard imposes additional constraints on character semantics and transmittability, not all implementations that are compliant with ISO/IEC 10646 will be compliant with the Unicode Standard.

C.5 Identification of Features for Unicode

ISO/IEC 10646 provides mechanisms for specifying a number of implementation parameters. ISO/IEC 10646 contains no means of explicitly declaring the Unicode Standard as such. As a whole, however, the Unicode Standard may be considered as encompassing the entire repertoire of ISO/IEC 10646 and having the following features (as well as additional semantics):

- Encoding forms: UTF-8, UTF-16, or UTF-32

- Encoding schemes: UTF-8, UTF-16BE, UTF-16LE, UTF-16, UTF-32BE, UTF-32LE, or UTF-32

Few applications are expected to make use of all of the characters defined in ISO/IEC 10646. The conformance clauses of the two standards address this situation in very different ways. ISO/IEC 10646 provides a mechanism for specifying included subsets of the character repertoire, permitting implementations to ignore characters that are not included (see normative Annex A of ISO/IEC 10646). A Unicode implementation requires a minimal level of handling all character codes—namely, the ability to store and retransmit them undamaged. Thus the Unicode Standard encompasses the entire ISO/IEC 10646 repertoire without requiring that any particular subset be implemented.

The Unicode Standard does not provide formal mechanisms for identifying a stream of bytes as Unicode characters, although to some extent this function is served by use of the *byte order mark* (U+FEFF) to indicate byte ordering. ISO/IEC 10646 defines an ISO/IEC 2022 control sequence to introduce the use of 10646. ISO/IEC 10646 also allows the use of U+FEFF as a "signature" as described in ISO/IEC 10646. This optional "signature" convention for identification of UTF-8, UTF-16, and UTF-32 is described in the informative Annex H of 10646. It is consistent with the description of the *byte order mark* in *Section 23.8, Specials*.

C.6 Character Names

Unicode character names follow the ISO/IEC character naming guidelines (summarized in informative Annex L of ISO/IEC 10646). In the first version of the Unicode Standard, the naming convention followed the ISO/IEC naming convention, but with some differences that were largely editorial. For example,

ISO/IEC 10646 name 029A LATIN SMALL LETTER CLOSED OPEN E
Unicode 1.0 name 029A LATIN SMALL LETTER CLOSED EPSILON

In the ISO/IEC framework, the unique character name is viewed as the major resource for both character semantics and cross-mapping among standards. In the framework of the Unicode Standard, character semantics are indicated via character properties, functional specifications, usage annotations, and name aliases; cross-mappings among standards are provided in the form of explicit tables available on the Unicode website. The disparities between the Unicode 1.0 names and ISO/IEC 10646 names have been remedied by adoption of ISO/IEC 10646 names in the Unicode Standard. The names adopted by the Unicode Standard are from the English-language version of ISO/IEC 10646, even when other language versions are published by ISO.

C.7 Character Functional Specifications

The core of a character code standard is a mapping of code points to characters, but in some cases the semantics or even the identity of the character may be unclear. Certainly a character is not simply the representative glyph used to depict it in the standard. For this reason, the Unicode Standard supplies the information necessary to specify the semantics of the characters it encodes.

Thus the Unicode Standard encompasses far more than a chart of code points. It also contains a set of extensive character functional specifications and data, as well as substantial background material designed to help implementers better understand how the characters interact. The Unicode Standard specifies properties and algorithms. Conformant implementations of the Unicode Standard will also be conformant with ISO/IEC 10646.

Compliant implementations of ISO/IEC 10646 can be conformant to the Unicode Standard—as long as the implementations conform to all additional specifications that apply to the characters of their adopted subsets, and as long as they support all Unicode characters outside their adopted subsets in the manner referred to in *Appendix C.5, Identification of Features for Unicode*.

Appendix D

Version History of the Standard

This appendix provides version history of the standard. Updates to data files are documented in Unicode Standard Annex #44, "Unicode Character Database."

The Unicode Technical Committee updates the Unicode Standard to respond to the needs of implementers and users while maintaining consistency with ISO/IEC 10646. The relationship between these versions of Unicode and ISO/IEC 10646 is shown in *Table D-1*. For more detail on the relationship of Unicode and ISO/IEC 10646, see *Appendix C, Relationship to ISO/IEC 10646*.

Table D-1. Versions of Unicode and ISO/IEC 10646

Year	Version	Published	ISO/IEC 10646
1991	Unicode 1.0	Vol. 1, Addison-Wesley	Basis for Committee Draft 2 of 10646-1
1992	Unicode 1.0.1	Vol. 1, 2, Addison-Wesley	Interim merger version
1993	Unicode 1.1	UTR #4	Matches 10646-1
1996	Unicode 2.0	Addison-Wesley	Matches 10646-1 plus amendments
1998	Unicode 2.1	UTR #8	Matches 10646-1 plus amendments
2000	Unicode 3.0	Addison-Wesley	Matches 10646-1 second edition
2001	Unicode 3.1	UAX #27	Matches 10646-1 second edition plus two characters, 10646-2 first edition
2002	Unicode 3.2	UAX #28	Matches 10646-1 second edition plus amendment, 10646-2 first edition
2003	Unicode 4.0	Addison-Wesley	Matches 10646:2003, third version
2005	Unicode 4.1	Web publication	Matches 10646:2003, third version, plus Amd 1
2006	Unicode 5.0	Addison-Wesley (2007)	Matches 10646:2003, third version, plus Amd 1, Amd 2, and four characters from Amd 3
2008	Unicode 5.1	Web publication	Matches 10646:2003, third version, plus Amd 1 through Amd 4
2009	Unicode 5.2	Web publication	Matches 10646:2003, third version, plus Amd 1 through Amd 6
2010	Unicode 6.0	Web publication	Matches 10646:2011, second edition
2012	Unicode 6.1	Web publication	Matches 10646:2012, third edition
2012	Unicode 6.2	Web publication	Matches 10646:2012, third edition

2013	Unicode 6.3	Web publication	Matches 10646:2012, third edition
2014	Unicode 7.0	Web publication	Matches 10646:2012, third edition, plus Amd 1 and Amd 2
2015	Unicode 8.0	Web publication	Matches 10646:2014, fourth edition, plus Amd 1, and 51 characters from Amd 2
2016	Unicode 9.0	Web publication	Matches 10646:2014, fourth edition, plus Amd 1 and Amd 2, and 273 characters from 10646, fifth edition
2017	Unicode 10.0	Web publication	Matches 10646:2017, fifth edition, plus 344 characters from Amd 1
2018	Unicode 11.0	Web publication	Matches 10646:2017, fifth edition, plus Amd 1, and 117 characters from Amd 2.
2019	Unicode 12.0	Web publication	Matches 10646:2017, fifth edition, plus Amd 1 and Amd 2, and 62 characters from 10646, sixth edition
2020	Unicode 13.0	Web publication	Matches 10646:2020, sixth edition
2021	Unicode 14.0	Web publication	Adds 838 characters
2022	Unicode 15.0	Web publication	Matches 10646:2020, sixth edition, plus Amd 1
2023	Unicode 15.1	Web publication	Adds 627 characters
2024	Unicode 16.0	Web publication	Matches 10646:2020, sixth edition, plus Amd 1 and Amd 2

Appendix E

Han Unification History

Efforts to standardize a comprehensive Han character repertoire go back at least as far as the Eastern Han dynasty, when the important dictionary *Shuowen Jiezi* (121 CE) codified a set of some 10,000 characters and variants, crystallizing earlier Qin dynasty initiatives at orthographic reform. Subsequent dictionaries in China grew larger as each generation re-combined the *Shuowen* script elements to create new characters. By the time the Qing dynasty *Kang Xi* dictionary was completed in the 18th century, the character set had grown to include more than 40,000 characters and variants. In relatively recent times many more characters and variants have been created and catalogued, reflecting modern PRC simplification and standardization initiatives, as well as ongoing inventories of legacy printed texts.

The effort to create a unified Han character encoding was guided by the developing national standards, driven by offshoots of the dictionary traditions just mentioned, and focused on modern bibliographic and pedagogical lists of characters in common use in various genres. Much of the early work to create national and transnational encoding standards was published in China and Japan in the late 1970s and early 1980s.

The Chinese Character Code for Information Interchange (CCCII), first published in Taiwan in 1980, identified a set of some 5,000 characters in frequent use in China, Taiwan, and Japan. (Subsequent revisions of CCCII considerably expanded the set.) In somewhat modified form, CCCII was adopted for use in the United States as ANSI Z39.64-1989, also known as EACC, the *East Asian Character Code For Bibliographic Use*. EACC encoded some 16,000 characters and variants, organized using a twelve-layer variant mapping mechanism.

In 1980, Takahashi Tokutaro of Japan's National Diet Library proposed ISO standard-ization of a character set for common use among East Asian countries. This proposal included a report on the first *Japanese Industrial Standard* for *kanji* coding (JIS C 6226-1978). Published in January 1978, JIS C 6226-1978 was growing in influence: it encoded a total of 6,349 *kanji* arranged in two levels according to frequency of use, and approximately 500 other characters, including Greek and Cyrillic.

E.1 Development of the URO

The Unicode Han character set began with a project to create a Han character cross-reference database at Xerox in 1986. In 1988, a parallel effort began at Apple based on the RLG's CJK Thesaurus, which is used to maintain EACC. The merger of the Apple and Xerox databases in 1989 led to the first draft of the Unicode Han character set. At the September 1989 meeting of X3L2 (an accredited standards committee for codes and character sets operating under the procedures of the American National Standards Institute), the Unicode Working Group proposed this set for inclusion in ISO/IEC 10646.

The primary difference between the Unicode Han character repertoire and earlier efforts was that the Unicode Han character set extended the bibliographic sets to guarantee complete coverage of industry and newer national standards. The unification criteria employed in this original Unicode Han character repertoire were based on rules used by JIS and on a set of Han character identity principles (*rentong yuanze*) being developed in China by experts working with the Association for a Common Chinese Code (ACCC). An important principle was to preserve all character distinctions within existing and proposed national and industry standards.

The Unicode Han proposal stimulated interest in a unified Han set for inclusion in ISO/IEC 10646, which led to an ad hoc meeting to discuss the issue of unification. Held in Beijing in October 1989, this meeting was the beginning of informal cooperation between the Unicode Working Group and the ACCC to exchange information on each group's proposals for Han unification.

A second ad hoc meeting on Han unification was held in Seoul in February 1990. At this meeting, the Korean delegation proposed the establishment of a group composed of the East Asian countries and other interested organizations to study a unified Han encoding. From this informal meeting emerged the Chinese/Japanese/Korean Joint Research Group (hereafter referred to as the CJK-JRG).

A second draft of the Unicode Han character repertoire was sent out for widespread review in December 1990 to coincide with the announcement of the formation of the Unicode Consortium. The December 1990 draft of the Unicode Han character set differed from the first draft in that it used the principle of *Kang Xi* radical-stroke ordering of the characters. To verify independently the soundness and accuracy of the unification, the Consortium arranged to have this draft reviewed in detail by East Asian scholars at the University of Toronto.

In the meantime, China announced that it was about to complete its own proposal for a Han Character Set, GB 13000. Concluding that the two drafts were similar in content and philosophy, the Unicode Consortium and the Center for Computer and Information Development Research, Ministry of Machinery and Electronic Industry

(CCID, China's computer standards body), agreed to merge the two efforts into a single proposal. Each added missing characters from the other set and agreed upon a method for ordering the characters using the four-dictionary ordering scheme described in *Section 18.1, Han*. Both proposals benefited greatly from programmatic comparisons of the two databases.

As a result of the agreement to merge the Unicode Standard and ISO/IEC 10646, the Unicode Consortium agreed to adopt the unified Han character repertoire that was to be developed by the CJK-JRG.

The first CJK-JRG meeting was held in Tokyo in July 1991. The group recognized that there was a compelling requirement for unification of the existing CJK ideographic characters into one coherent coding standard. Two basic decisions were made: to use GB 13000 (previously merged with the Unicode Han repertoire) as the basis for what would be termed "The Unified Repertoire and Ordering," and to verify the unification results based on rules that had been developed by Professor Miyazawa Akira and other members of the Japanese delegation.

The formal review of GB 13000 began immediately. Subsequent meetings were held in Beijing and Hong Kong. On March 27, 1992, the CJK-JRG completed the *Unified Repertoire and Ordering* (URO)*, Version 2.0*. This repertoire was subsequently published both by the Unicode Consortium in *The Unicode Standard, Version 1.0*, Volume 2, and by ISO in ISO/IEC 10646-1:1993.

E.2 Continuing Research on Ideographs

E.2.1 Ideographic Rapporteur Group

In October 1993, the CJK-JRG became a formal subgroup of ISO/IEC JTC1/SC2/WG2 and was renamed the Ideographic Rapporteur Group (IRG). The IRG now has the formal responsibility of developing extensions to the URO 2.0 to expand the encoded repertoire of unified CJK ideographs. The Unicode Consortium participates in this group as a liaison member of ISO.

In its second meeting held in Hà Nội in February 1994, the IRG agreed to include Vietnamese Chữ Nôm ideographs in a future version of the URO and to add a fifth reference dictionary to the ordering scheme.

In 1998, the IRG completed work on the first ideographic supplement to the URO, CJK Unified Ideographs Extension A. This set of 6,582 characters was culled from national and industrial standards and historical literature and was first encoded in *The Unicode Standard, Version 3.0*. CJK Unified Ideographs Extension A represents the final set of CJK unified ideographs to be encoded on the BMP.

In 2000, the IRG completed work on the second ideographic supplement to the URO, a very large collection known as CJK Unified Ideographs Extension B. These 42,711 characters were derived from major classical dictionaries and literary sources, and from many additional national standards, as documented in *Appendix E.3, CJK Sources*. The Extension B block was first encoded in *The Unicode Standard, Version 3.1*, and is the first block of CJK unified ideographs to be encoded on Plane 2.

In 2005, the IRG identified a subset of the unified CJK ideographs, called the International Ideograph Core (IICore). This subset is designed to serve as a relatively small block of around 10,000 ideographs, mainly for use in devices with limited resources, such as mobile phones. The IICore subset is meant to cover the vast majority of modern texts in all regions where ideographs are used. The repertoire of the IICore subset is identified with the kIICore property in the Unihan Database. A related but much larger set, UnihanCore2020, was standardized by the Unicode Technical Committee in 2020 (see "UnihanCore2020" in *Section 18.1, Han*).

Also in 2005, a small set of ideographs was encoded to support the complete repertoire of the GB 18030-2000 and HKSCS-2004 standards. In addition, an initial set of CJK strokes was encoded.

In 2008, the IRG completed work on the third ideographic supplement to the URO, a block of 4,149 characters from various sources. The Extension C block was first encoded in *The Unicode Standard, Version 5.2*.

In 2009, the IRG completed work on the fourth ideographic supplement to the URO, a block of 222 characters from various sources as documented in *Appendix E.3, CJK Sources*. The Extension D block represents a small number of characters which IRG member bodies felt were urgently needed. The extension D block was first encoded in *The Unicode Standard, Version 6.0*.

In 2012, the IRG completed work on the fifth supplement to the URO, a block of 5,762 characters from various sources. The Extension E block was first encoded in *The Unicode Standard, Version 8.0*.

In 2015, the IRG completed work on the sixth supplement to the URO, a block of 7,473 characters from various sources. The Extension F block was first encoded in *The Unicode Standard, Version 10.0*.

E.2.2 Ideographic Research Group

In June 2019, the Ideographic Rapporteur Group was renamed to Ideographic Research Group, preserving the same "IRG" acronym.

In 2019, the IRG completed work on the seventh supplement to the URO, a block of 4,939 characters from various sources. The Extension G block was first encoded in *The Unicode Standard, Version 13.0*, and is the first collection of CJK unified ideographs to be encoded on Plane 3.

In 2021, the IRG completed work on the eighth supplement to the URO, a block of 4,192 characters from various sources. The Extension H block was first encoded in *The Unicode Standard, Version 15.0*, and is the second collection of CJK unified ideographs to be encoded on Plane 3.

The IRG received submissions for what was originally intended to become the ninth supplement from member bodies in mid-2021, which included submissions from China, SAT, South Korea, TCA, the United Kingdom, the Unicode Consortium, and Vietnam. This supplement is now expected to become the Extension J block.

In 2023, the UTC and several individual IRG experts, in response to a draft amendment of China's GB 18030-2022 standard that included several hundred urgently needed characters, worked with China to develop the ninth supplement to the URO, a block of 622 urgently needed characters from China. The Extension I block was first encoded in *The Unicode Standard, Version 15.1*, and immediately follows the Extension F block in Plane 2.

E.3 CJK Sources

The Unicode Standard draws its unified Han character repertoire from a number of different character set standards. These standards, dictionaries and other documents are grouped into 11 sources. The detailed listing of all of those sources, including bibliographic references for the various standards and other documents involved, can be found in Unicode Standard Annex #38, "Unicode Han Database (Unihan)." The primary work of unifying and ordering the characters from these sources was done by the Ideographic Research Group (IRG).

The G, T, H, M, J, K, KP, S, U, UK, and V sources represent the characters submitted to the IRG by its member bodies. The G source consists of submissions from the People's Republic of China and Singapore. The other ten sources are the submissions from TCA (Taipei Computer Association), Hong Kong SAR, Macao SAR, Japan, South Korea, North Korea, SAT (Saṃgaṇikīkṛtaṃ Taiśotripiṭakaṃ) Daizōkyō Text Database Committee, the Unicode Consortium, the United Kingdom, and Vietnam, respectively.

Omission of Repertoire for Some Sources. In some cases, the entire ideographic repertoire of the original character set standards was *not* included in the corresponding source. Three reasons explain this decision:

1. Where the repertoires of two of the character set standards within a single source have considerable overlap, the characters in the overlap might be included only once in the source. This approach was used, for example, with GB/T 2312-1980 and GB/T 12345-1990, which have many ideographs in common. Characters in GB/T 12345-1990 that are duplicates of characters in GB/T 2312-1980 were not included in the G source.

2. Where a character set standard is based on unification rules that differ substantially from those used by the IRG, many variant characters found in the character set standard will not be included in the source. This situation is the case with CNS 11643-1992, EACC, and CCCII. It is the only case where full round-trip compatibility with the Han ideograph repertoire of the relevant character set standards is not guaranteed.

3. KS C 5601-1987 contains numerous duplicate ideographs included because they have multiple pronunciations in Korean. They are encoded in the CJK Compatibility Ideographs block to provide full round-trip compatibility with KS C 5601-1987 (now known as KS X 1001:2004).

Appendix F

Documentation of CJK Strokes

This appendix provides additional documentation regarding each of the CJK stroke characters encoded in the CJK Strokes block (U+31C0..U+31EF). For a general introduction to CJK characters and CJK strokes, see *Section 18.1, Han*.

The information in *Table F-1* gives five types of identifying data for each CJK stroke. Each stroke is also exemplified in a spanning lower row, with a varying number of examples, as appropriate. The information contained in each of the five columns and in the examples row is described more specifically below.

- **Stroke**:A representative glyph for each CJK stroke character, with its Unicode code point shown underneath.
- **Acronym**: The abbreviation used in the Unicode character name for the CJK stroke character.
- **Pīnyīn**: The Hanyu Pinyin (Modern Standard Chinese) romanization of the stroke name (as given in the next column).
- **Name**: A traditional name for this stroke, as written in Han characters.
- **Variant**: Alternative (context-specific) forms of the representative glyph for this stroke, if any.
- **Examples**: Representative glyphs and variant forms of CJK unified ideographs, exemplifying typical usage of this stroke type in Han characters. The stroke or strokes in question are highlighted in red to make it easier to spot them in the example glyphs. Each example glyph (or variant) is followed by the Unicode code point for the CJK unified ideograph character it represents, for easy reference.

The CJK stroke characters in the table are ordered according to the traditional "Five Types."

Table F-1. CJK Strokes

Stroke	Acronym	Pīnyīn	Name	Variant
筆畫	縮寫	拼音	筆畫名	異體

一 U+31D0	**H**	héng	橫	一
一 U+4E00, 二 U+4E8C, 三 U+4E09, 丁 U+4E01, 丞 U+4E1E, 丈 U+4E08, 世 U+4E16, 不 U+4E0D, 十 U+5341, 卅 U+5345, 目 U+76EE, 七 U+4E03, 毛 U+6BDB				
⼂ U+31C0	**T**	tí	提	⼂
扌 U+624C, 氵U+6C3A, 录 U+5F55, 暴 U+66B4, 地 U+5730, 虫 U+866B, 默 U+9EC8, 冫 U+51AB, 冶 U+51B6, 冽 U+51BD, 氵 U+6C35, 淋 U+6DCB, 治 U+6CBB				
│ U+31D1	**S**	shù	豎	│
│ U+4E28, ㄐ U+4E29, 上 U+4E0A, 工 U+5DE5, 中 U+4E2D, 串 U+4E32, 讠 U+8BA7, 乍 U+4E4D, 五 U+4E94, 丑 U+4E11				
┘ U+31DA	**SG**	shù gōu	豎鉤	┘
亅 U+4E85, 水 U+6C34, 才 U+624D, 示 U+793A, 爭 U+722D, 事 U+4E8B				
ノ U+31D2	**P**	piě	撇	ノ
丿 U+4E3F, 人 U+4EBA, 乂 U+4E42, 爻 U+723B, 須 U+9808, 俗 U+4FD7, 禾 U+79BE, 乏 U+4E4F, 乖 U+4E56, 采 U+91C6, 衣 U+8863, 行 U+884C				
⼃ U+31D3	**SP**	shù piě	豎撇	⼃
乃 U+4E43, 几 U+51E0, 大 U+5927, 厂 U+5382, 川 U+5DDD, 亦 U+4EA6, 用 U+7528, 屮 U+5C70, 判 U+5224, 州 U+5DDE, 班 U+73ED, 病 U+75C5, 齊 U+9F4A				
、 U+31D4	**D**	diǎn	點	丶
丶 U+4E36, 丸 U+4E38, 义 U+4E49, 永 U+6C38, 斗 U+6597, 冰 U+51B0, 凡 U+51E1, 丹 U+4E39, 主 U+4E3B, 求 U+6C42, 火 U+706B, 刀 U+5203, 灬 U+706C				

㇏ U+31CF	**N**	nà	捺	㇏ ㇏
人 U+4EBA, 天 U+5929, 入 (入入) U+5165, 尺 U+5C3A, 孑 U+5B53, 餐 U+9910, 尖 U+5C16, 走 U+8D70, 是 U+662F, 廷 U+5EF7, 麩 U+9EA9				
㇝ U+31DD	**TN**	tí nà	提捺	㇝ ㇝
㇝ U+4E40, 尐 U+5C10, 八 (八) U+516B, 入 (入) U+5165, 文 (文) U+6587, 夂 (夂) U+590A, 廻 (廻) U+5EFB, 之 U+4E4B, 返 U+8FD4, 逆 U+9006				
㇅ U+31D5	**HZ**	héng zhé	橫折	ㄱ
㇅ U+200CD, 口 U+53E3, 囗 U+56D7, 田 U+7530, 呂 U+5415, 品 U+54C1, 申 U+7533, 甲 U+7532, 圓 U+5706, 彐 U+5DEA, 今 (今) U+4ECA				
㇇ U+31C7	**HP**	héng piě	橫撇	㇇
又 U+53C8, 双 U+53CC, 叕 U+53D2, 今 (今) U+4ECA				
㇖ U+31D6	**HG**	héng gōu	橫鉤	㇖
㇖ U+4E5B, ㇖ U+5196, 了 U+4E86, 子 U+5B50, 予 U+4E88, 矛 U+77DB, 疋 U+758B, 写 U+5199, 字 U+5B57, 疏 U+758F				
㇄ U+31D7	**SZ**	shù zhé	豎折	ㄴ
㇄ U+200CA, ㇄ U+200CB, 山 U+5C71, 出 U+51FA, 牙 U+7259, 互 U+4E92, 乐 U+4E50, 东 U+4E1C, 毌 U+6BCC, 断 U+65AD, 繼 U+7E7C, 彙 U+5F59				
㇄ U+31C4	**SW**	shù wān	豎彎	
㐄 (㐄) U+5338, 區 (區) U+5340, 亡 (亡) U+4EA1, 妄 (妄) U+5984, 四 (四) U+56DB				

┘ U+31D8	**SWZ**	shù wān zuǒ	豎彎左	┘
┘ U+200CE, 帯 (帯) U+23D92, 肅 (肅) U+8085, 嘯 U+56t2F, 蕭 U+856D, 淵 U+6DF5					

↳ U+31D9	**ST**	shù tí	豎提	
↳ U+2010C, 民 U+6C11, 艮 U+826E, 良 U+826F, 食 U+98DF, 很 U+5F88, 鄉 U+9109				

ㄥ U+31DC	**PZ**	piě zhé	撇折	ㄥ
ㄙ U+53B6, 幺 U+5E7A, 糸 U+7CF8, 絲 U+7D72, 弘 U+5F18, 公 U+516C, 翁 U+7FC1				

ㄑ U+31DB	**PD**	piě diǎn	撇點	ㄑ
ㄑ U+21FE8, 巛 U+5DDB, 女 U+5973, 巡 U+5DE1, 災 U+707D, 甾 U+753E, 鼠 U+5DE4, 獵 U+7375, 綏 U+7D8F				

ノ U+31E2	**PG**	piě gōu	撇鈎	
乄 U+4E44				

) U+31C1	**WG**	wān gōu	彎鈎	
犭 U+72AD, 豕 U+8C55, 豸 U+8C78, 狐 U+72D0, 嶽 U+5DBD, 貓 U+8C93, 家 U+5BB6, 逐 U+9010				

し U+31C2	**XG**	xié gōu	斜鈎	
弋 U+5F0B, 戈 U+6208, 我 U+6211, 戰 U+6230, 鼬 U+9F2C				

し U+31C3	**BXG**	biǎn xié gōu	扁斜鈎	し し
心 心 U+5FC3 (楷 "心" Regular script)				

㇅ U+31C5	**HZZ**	héng zhé zhé	横折折	
凹 U+51F9, 卍 U+534D, 㙿 U+2067F				
㇍ U+31CD	**HZW**	héng zhé wān	横折彎	
几 U+20627, 殳 U+6BB3, 朵 U+6735, 投 U+6295				
㇊ U+31CA	**HZT**	héng zhé tí	横折提	
殼 U+6BBC, 鳩 U+9CE9, 讠 U+8BA0, 计 U+8BA1				
㇆ U+31C6	**HZG**	héng zhé gōu	横折鉤	㇆ ㇆
㇌ U+200CC, 习 U+4E60, 羽 U+7FBD, 包 U+5305, 勻 U+52FB, 葡 U+8461, 邪 U+90A3, 靑 U+9751, 甫 U+752B, 勺 U+52FA, 月 U+6708, 巳 U+4E5C, 也 U+4E5F				
㇤ U+31E4	**HXG**	héng xié gōu	横斜鉤	
飞 U+98DE, 风 U+98CE, 气 U+6C14, 虱 U+8671, 瘋 U+760B, 訊 U+8A0A				
㇞ U+31DE	**SZZ**	shù zhé zhé	豎折折	
㇑ U+200D1, 卐 U+5350, 亞 U+4E9E, 鼎 U+9F0E, 吳 U+5433				
㇥ U+31E5	**SZP**	shù zhé piě	豎折撇	㇥
�057 U+594A, 矢 U+5928, 专 U+4E13, 設 U+279AE, 卺 U+244F7, 㒡 U+249A1				
㇟ U+31DF	**SWG**	shù wān gōu	豎彎鉤	
乚 U+4E5A, 礼 U+793C, 乱 U+4E71, 儿 U+513F, 己 U+5DF1, 已 U+5DF2, 巳 U+5DF3, 心 U+5FC3, 必 U+5FC5				

㇎ U+31CE	**HZZZ**	héng zhé zhé zhé	橫折折折	
凸 U+51F8, 㸭 U+21E2D				
㇋ U+31CB	**HZZP**	héng zhé zhé piě	橫折折撇	
及 U+53CA, 廴 U+5EF4, 建 U+5EFA				
㇈ U+31C8	**HZWG**	héng zhé wān gōu	橫折彎鉤	
九 U+4E5D, 几 U+51E0, 丸 U+4E38, 旭 U+65ED, 杂 U+6742				
㇠ U+31E0	**HXWG**	héng xié wān gōu	橫斜彎鉤	
乙 U+4E59, 氹 U+6C39, 乞 U+4E5E				
㇌ U+31CC	**HPWG**	héng piě wān gōu	橫撇彎鉤	
阝 U+961D, 队 U+961F, 邮 U+90AE				
㇉ U+31C9	**SZWG**	shù zhé wān gōu	豎折彎鉤	㇉
丂 U+4E02, 与 U+4E0E, 亏 U+4E8F, 号 U+53F7, 弓 U+5F13, 强 U+5F3A, 姊 U+59CA, 函 U+51FD, 丐 U+4E10, 马 U+9A6C, 鸟 U+9E15				
㇡ U+31E1	**HZZZG**	héng zhé zhé zhé gōu	橫折折折鉤	
乃 U+2010E, 乃 U+4E43, 仍 U+4ECD, 孕 U+5B55, 𠃓 U+200D3, 汤 U+6C64				
㇣ U+31E3	**Q**	quān	圈	○
○ U+3007, 㔔 U+3514, 㪳 U+3AB3, 㫈 U+3AC8				