

The Unicode Standard

Version 6.1 – Core Specification

The Unicode Consortium

Edited by

Julie D. Allen, Deborah Anderson, Joe Becker, Richard Cook,
Mark Davis, Peter Edberg, Michael Everson, Asmus Freytag,
John H. Jenkins, Rick McGowan, Lisa Moore, Eric Muller,
Addison Phillips, Michel Suignard, and Ken Whistler

 Unicode Consortium

Mountain View, CA

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc., in the United States and other countries.

The authors and publisher have taken care in the preparation of this specification, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The *Unicode Character Database* and other files are provided as-is by Unicode, Inc. No claims are made as to fitness for any particular purpose. No warranties of any kind are expressed or implied. The recipient agrees to determine applicability of information provided.

Copyright © 1991–2012 Unicode, Inc.

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction. For information regarding permissions, inquire at <http://www.unicode.org/reporting.html>. For information about the Unicode terms of use, please see <http://www.unicode.org/copyright.html>.

The Unicode Standard / the Unicode Consortium ; edited by Julie D. Allen ... [et al.]. — Version 6.1.

Includes bibliographical references and index.

ISBN 978-1-936213-02-3 (<http://www.unicode.org/versions/Unicode6.1.0/>)

1. Unicode (Computer character set) I. Allen, Julie D. II. Unicode Consortium.

QA268.U545 2012

ISBN 978-1-936213-02-3

Published in Mountain View, CA

April 2012

Contents

Figures	xvii
Tables	xxi
Preface	xxv
Why Unicode?	xxv
What's New?	xxv
Organization of This Standard	xxvi
Unicode Standard Annexes	xxvii
The Unicode Character Database	xxviii
Unicode Code Charts	xxviii
Unicode Technical Standards and Unicode Technical Reports	xxviii
Updates and Errata	xxix
Acknowledgements	xxix
1 Introduction	1
1.1 Coverage	2
Standards Coverage	2
New Characters	3
1.2 Design Goals	3
1.3 Text Handling	4
Characters and Glyphs	5
Text Elements	5
2 General Structure	7
2.1 Architectural Context	7
Basic Text Processes	8
Text Elements, Characters, and Text Processes	8
Text Processes and Encoding	9
2.2 Unicode Design Principles	10
Universality	10
Efficiency	11
Characters, Not Glyphs	11
Semantics	14
Plain Text	14
Logical Order	15
Unification	17
Dynamic Composition	18
Stability	18
Convertibility	19
2.3 Compatibility Characters	19
Compatibility Variants	20
Compatibility Decomposable Characters	20
2.4 Code Points and Characters	21
Types of Code Points	22
2.5 Encoding Forms	24
UTF-32	26
UTF-16	27
UTF-8	27
Comparison of the Advantages of UTF-32, UTF-16, and UTF-8	28

2.6	Encoding Schemes	30
2.7	Unicode Strings	32
2.8	Unicode Allocation	33
	Planes	33
	Allocation Areas and Character Blocks	34
	Assignment of Code Points	35
2.9	Details of Allocation	35
	Plane 0 (BMP)	37
	Plane 1 (SMP)	38
	Plane 2 (SIP)	40
	Other Planes	40
2.10	Writing Direction	40
2.11	Combining Characters	41
	Sequence of Base Characters and Diacritics	42
	Multiple Combining Characters	43
	Ligated Multiple Base Characters	45
	Exhibiting Nonspacing Marks in Isolation	45
	“Characters” and Grapheme Clusters	46
2.12	Equivalent Sequences and Normalization	46
	Normalization	47
	Decompositions	48
	Non-decomposition of Overlaid Diacritics	49
2.13	Special Characters and Noncharacters	50
	Special Noncharacter Code Points	50
	Byte Order Mark (BOM)	50
	Layout and Format Control Characters	51
	The Replacement Character	51
	Control Codes	51
2.14	Conforming to the Unicode Standard	51
	Characteristics of Conformant Implementations	52
	Unacceptable Behavior	52
	Acceptable Behavior	53
	Supported Subsets	53
3	Conformance	55
3.1	Versions of the Unicode Standard	55
	Stability	56
	Version Numbering	56
	Errata and Corrigenda	57
	References to the Unicode Standard	57
	Precision in Version Citation	57
	References to Unicode Character Properties	58
	References to Unicode Algorithms	58
3.2	Conformance Requirements	59
	Code Points Unassigned to Abstract Characters	59
	Interpretation	59
	Modification	61
	Character Encoding Forms	62
	Character Encoding Schemes	62
	Bidirectional Text	63
	Normalization Forms	63
	Normative References	63
	Unicode Algorithms	63

	Default Casing Algorithms	64
	Unicode Standard Annexes	64
3.3	Semantics	64
	Definitions	64
	Character Identity and Semantics	65
3.4	Characters and Encoding	66
3.5	Properties	70
	Types of Properties	70
	Property Values	71
	Default Property Values	72
	Classification of Properties by Their Values	72
	Property Status	73
	Context Dependence	76
	Stability of Properties	76
	Simple and Derived Properties	77
	Property Aliases	77
	Private Use	78
3.6	Combination	79
	Combining Character Sequences	79
	Grapheme Clusters	81
	Application of Combining Marks	82
3.7	Decomposition	86
	Compatibility Decomposition	87
	Canonical Decomposition	88
3.8	Surrogates	88
3.9	Unicode Encoding Forms	89
	UTF-32	93
	UTF-16	93
	UTF-8	94
	Encoding Form Conversion	95
	Constraints on Conversion Processes	95
3.10	Unicode Encoding Schemes	97
3.11	Normalization Forms	101
	Normalization Stability	101
	Combining Classes	102
	Specification of Unicode Normalization Forms	103
	Starters	103
	Canonical Ordering Algorithm	104
	Canonical Composition Algorithm	104
	Definition of Normalization Forms	106
3.12	Conjoining Jamo Behavior	107
	Definitions	107
	Hangul Syllable Decomposition	109
	Hangul Syllable Composition	110
	Hangul Syllable Name Generation	111
	Sample Code for Hangul Algorithms	112
3.13	Default Case Algorithms	115
	Definitions	115
	Default Case Conversion	117
	Default Case Folding	117
	Default Case Detection	118
	Default Caseless Matching	119

4	Character Properties	121
4.1	Unicode Character Database	122
4.2	Case	124
	Definitions of Case and Casing	124
	Case Mapping	126
4.3	Combining Classes	126
	Reordrant, Split, and Subjoined Combining Marks	127
4.4	Directionality	130
4.5	General Category	130
4.6	Numeric Value	133
	Ideographic Numeric Values	133
4.7	Bidi Mirrored	135
4.8	Name	135
	Unicode Name Property	137
	Code Point Labels	138
	Use of Character Names in APIs and User Interfaces	138
4.9	Unicode 1.0 Names	139
4.10	Letters, Alphabetic, and Ideographic	139
4.11	Properties Related to Text Boundaries	140
4.12	Characters with Unusual Properties	140
5	Implementation Guidelines	145
5.1	Data Structures for Character Conversion	145
	Issues	145
	Multistage Tables	146
5.2	Programming Languages and Data Types	147
	Unicode Data Types for C	147
5.3	Unknown and Missing Characters	148
5.4	Handling Surrogate Pairs in UTF-16	149
5.5	Handling Numbers	151
5.6	Normalization	152
5.7	Compression	153
5.8	Newline Guidelines	153
	Definitions	153
	Line Separator and Paragraph Separator	154
	Recommendations	155
5.9	Regular Expressions	156
5.10	Language Information in Plain Text	157
	Requirements for Language Tagging	157
	Language Tags and Han Unification	157
5.11	Editing and Selection	158
	Consistent Text Elements	158
5.12	Strategies for Handling Nonspacing Marks	159
	Keyboard Input	160
	Truncation	161
5.13	Rendering Nonspacing Marks	162
	Canonical Equivalence	164
	Positioning Methods	165
5.14	Locating Text Element Boundaries	167
5.15	Identifiers	167

5.16	Sorting and Searching	167
	Culturally Expected Sorting and Searching	168
	Language-Insensitive Sorting	168
	Searching	168
	Sublinear Searching	169
5.17	Binary Order	170
	UTF-8 in UTF-16 Order	170
	UTF-16 in UTF-8 Order	171
5.18	Case Mappings	172
	Titlecasing	172
	Complications for Case Mapping	172
	Reversibility	174
	Caseless Matching	175
	Normalization and Casing	177
5.19	Mapping Compatibility Variants	177
5.20	Unicode Security	179
5.21	Ignoring Characters in Processing	180
	Characters Ignored in Text Segmentation	181
	Characters Ignored in Line Breaking	181
	Characters Ignored in Cursive Joining	181
	Characters Ignored in Identifiers	182
	Characters Ignored in Searching and Sorting	182
	Characters Ignored for Display	183
5.22	Best Practice for U+FFFD Substitution	185
6	Writing Systems and Punctuation	187
6.1	Writing Systems	188
6.2	General Punctuation	191
	Blocks Devoted to Punctuation	193
	Format Control Characters	193
	Space Characters	194
	Dashes and Hyphens	195
	Paired Punctuation	197
	Language-Based Usage of Quotation Marks	197
	Apostrophes	200
	Other Punctuation	200
	Archaic Punctuation and Editorial Marks	204
	Indic Punctuation	206
	CJK Punctuation	207
	Unknown or Unavailable Ideographs	208
	CJK Compatibility Forms	209
7	European Alphabetic Scripts	211
7.1	Latin	212
	Letters of Basic Latin: U+0041–U+007A	215
	Letters of the Latin-1 Supplement: U+00C0–U+00FF	215
	Latin Extended-A: U+0100–U+017F	216
	Latin Extended-B: U+0180–U+024F	216
	IPA Extensions: U+0250–U+02AF	218
	Phonetic Extensions: U+1D00–U+1DBF	219
	Latin Extended Additional: U+1E00–U+1EFF	220
	Latin Extended-C: U+2C60–U+2C7F	220
	Latin Extended-D: U+A720–U+A7FF	221
	Latin Ligatures: U+FB00–U+FB06	222

7.2	Greek	222
	Greek: U+0370–U+03FF	222
	Greek Extended: U+1F00–U+1FFF	225
	Ancient Greek Numbers: U+10140–U+1018F	226
7.3	Coptic	227
7.4	Cyrillic	229
	Cyrillic: U+0400–U+04FF	230
	Cyrillic Supplement: U+0500–U+052F	230
	Cyrillic Extended-A: U+2DE0–U+2DFF	231
	Cyrillic Extended-B: U+A640–U+A69F	231
7.5	Glagolitic	231
7.6	Armenian	232
7.7	Georgian	233
7.8	Modifier Letters	235
	Spacing Modifier Letters: U+02B0–U+02FF	236
	Modifier Tone Letters: U+A700–U+A71F	237
7.9	Combining Marks	238
	Combining Diacritical Marks: U+0300–U+036F	242
	Combining Diacritical Marks Supplement: U+1DC0–U+1DFF	242
	Combining Marks for Symbols: U+20D0–U+20FF	242
	Combining Half Marks: U+FE20–U+FE2F	243
	Combining Marks in Other Blocks	243
8	Middle Eastern Scripts	245
8.1	Hebrew	246
	Hebrew: U+0590–U+05FF	246
	Alphabetic Presentation Forms: U+FB1D–U+FB4F	250
8.2	Arabic	250
	Arabic: U+0600–U+06FF	250
	Arabic Cursive Joining	256
	Arabic Ligatures	258
	Arabic Joining Groups	259
	Arabic Supplement: U+0750–U+077F	265
	Arabic Extended-A: U+08A0–U+08FF	265
	Arabic Presentation Forms-A: U+FB50–U+FDFF	265
	Arabic Presentation Forms-B: U+FE70–U+FEFF	266
8.3	Syriac	266
	Syriac: U+0700–U+074F	266
	Syriac Shaping	269
8.4	Samaritan	273
8.5	Thaana	274
9	South Asian Scripts-I	277
9.1	Devanagari	278
	Devanagari: U+0900–U+097F	278
	Principles of the Devanagari Script	279
	Rendering Devanagari	284
	Devanagari Digits, Punctuation, and Symbols	291
	Extensions in the Main Devanagari Block	292
	Devanagari Extended: U+A8E0–U+A8FF	293
	Vedic Extensions: U+1CD0–U+1CFF	294
9.2	Bengali (Bangla)	295
9.3	Gurmukhi	300

9.4	Gujarati	303
9.5	Oriya	304
9.6	Tamil	306
	Tamil: U+0B80–U+0BFF	306
	Tamil Vowels	307
	Tamil Ligatures	308
	Tamil Named Character Sequences	311
9.7	Telugu	313
9.8	Kannada	315
	Kannada: U+0C80–U+0CFF	315
	Principles of the Kannada Script	315
	Rendering Kannada	317
9.9	Malayalam	317
10	South Asian Scripts-II	323
10.1	Sinhala	324
10.2	Tibetan	325
10.3	Lepcha	335
10.4	Phags-pa	336
10.5	Limbu	342
10.6	Syloti Nagri	344
10.7	Kaithi	345
10.8	Saurashtra	347
10.9	Sharada	348
10.10	Takri	349
10.11	Chakma	350
10.12	Meetei Mayek	351
10.13	Ol Chiki	353
10.14	Sora Sompeng	354
10.15	Kharoshthi	355
	Kharoshthi: U+10A00–U+10A5F	355
	Rendering Kharoshthi	356
10.16	Brahmi	359
11	Southeast Asian Scripts	363
11.1	Thai	364
11.2	Lao	366
11.3	Myanmar	368
	Myanmar: U+1000–U+109F	368
	Myanmar Extended-A: U+AA60–U+AA7F	371
	Khamti Shan	372
	Aiton and Phake	373
11.4	Khmer	374
	Khmer: U+1780–U+17FF	374
	Principles of the Khmer Script	374
	Khmer Symbols: U+19E0–U+19FF	383
11.5	Tai Le	383
11.6	New Tai Lue	384
11.7	Tai Tham	385
11.8	Tai Viet	387

11.9 Kayah Li	389
11.10 Cham	390
11.11 Philippine Scripts	392
Tagalog: U+1700–U+171F	392
Hanunóo: U+1720–U+173F	392
Buhid: U+1740–U+175F	392
Tagbanwa: U+1760–U+177F	392
Principles of the Philippine Scripts	392
11.12 Buginese	393
11.13 Balinese	394
11.14 Javanese	399
11.15 Rejang	401
11.16 Batak	402
11.17 Sundanese	403
12 East Asian Scripts	405
12.1 Han	406
CJK Unified Ideographs	406
CJK Standards	407
Blocks Containing Han Ideographs	409
General Characteristics of Han Ideographs	411
Principles of Han Unification	414
Unification Rules	415
Abstract Shape	416
Han Ideograph Arrangement	418
Radical-Stroke Indices	419
Mappings for Han Ideographs	419
CJK Unified Ideographs Extension B: U+20000–U+2A6D6	420
CJK Unified Ideographs Extension C: U+2A700–U+2B734	420
CJK Unified Ideographs Extension D: U+2B740–U+2B81D	420
CJK Compatibility Ideographs: U+F900–U+FAFF	420
CJK Compatibility Supplement: U+2F800–U+2FA1D	421
Kanbun: U+3190–U+319F	421
Symbols Derived from Han Ideographs	421
CJK and KangXi Radicals: U+2E80–U+2FD5	421
CJK Additions from HKSCS and GB 18030	422
CJK Strokes: U+31C0–U+31EF	423
12.2 Ideographic Description Characters	423
12.3 Bopomofo	426
12.4 Hiragana and Katakana	428
Hiragana: U+3040–U+309F	428
Katakana: U+30A0–U+30FF	428
Katakana Phonetic Extensions: U+31F0–U+31FF	429
Kana Supplement U+1B000–U+1B0FF	429
12.5 Halfwidth and Fullwidth Forms	429
12.6 Hangul	430
Hangul Jamo: U+1100–U+11FF	430
Hangul Jamo Extended-A: U+A960–U+A97F	431
Hangul Jamo Extended-B: U+D7B0–U+D7FF	431
Hangul Compatibility Jamo: U+3130–U+318F	431
Hangul Syllables: U+AC00–U+D7A3	432
12.7 Yi	433

13 Additional Modern Scripts	437
13.1 Ethiopic	438
Ethiopic: U+1200–U+137F	438
Ethiopic Extensions	440
13.2 Mongolian	440
13.3 Osmanya	447
13.4 Tifinagh	448
13.5 N’Ko	450
13.6 Vai	454
13.7 Bamum	455
Bamum: U+A6A0–U+A6FF	455
Bamum Supplement: U+16800–U+16A3F	456
13.8 Cherokee	456
13.9 Canadian Aboriginal Syllabics	457
Canadian Aboriginal Syllabics: U+1400–U+167F	457
Canadian Aboriginal Syllabics Extended: U+18B0–U+18FF	458
13.10 Deseret	459
13.11 Shavian	461
13.12 Lisu	461
13.13 Miao	463
14 Additional Ancient and Historic Scripts	465
14.1 Ogham	466
14.2 Old Italic	467
14.3 Runic	469
14.4 Gothic	471
14.5 Old Turkic	472
14.6 Linear B	473
Linear B Syllabary: U+10000–U+1007F	473
Linear B Ideograms: U+10080–U+100FF	473
Aegean Numbers: U+10100–U+1013F	473
14.7 Cypriot Syllabary	474
14.8 Ancient Anatolian Alphabets	474
Lycian: U+10280–U+1029F	474
Carian: U+102A0–U+102DF	474
Lydian: U+10920–U+1093F	474
14.9 Old South Arabian	475
14.10 Phoenician	477
14.11 Imperial Aramaic	478
14.12 Mandaic	479
14.13 Inscriptional Parthian and Inscriptional Pahlavi	481
14.14 Avestan	482
14.15 Ugaritic	483
14.16 Old Persian	484
14.17 Sumero-Akkadian	485
Cuneiform: U+12000–U+123FF	485
Cuneiform Numbers and Punctuation: U+12400–U+1247F	487
14.18 Egyptian Hieroglyphs	487
14.19 Meroitic Hieroglyphs and Meroitic Cursive	491

15 Symbols	493
15.1 Currency Symbols	494
15.2 Letterlike Symbols	496
Letterlike Symbols: U+2100–U+214F	496
Mathematical Alphanumeric Symbols: U+1D400–U+1D7FF	497
Mathematical Alphabets	498
Fonts Used for Mathematical Alphabets	500
Arabic Mathematical Alphabetic Symbols: U+1EE00–U+1EEFF	501
15.3 Numerals	502
Decimal Digits	502
Other Digits	504
Non-Decimal Radix Systems	505
Acrophonic Systems and Other Letter-based Numbers	506
Rumi Numeral Forms: U+10E60–U+10E7E	507
CJK Numerals	507
Fractions	508
Common Indic Number Forms: U+A830–U+A83F	509
15.4 Superscript and Subscript Symbols	510
Superscripts and Subscripts: U+2070–U+209F	510
15.5 Mathematical Symbols	511
Mathematical Operators: U+2200–U+22FF	512
Supplements to Mathematical Symbols and Arrows	513
Supplemental Mathematical Operators: U+2A00–U+2AFF	514
Miscellaneous Mathematical Symbols-A: U+27C0–U+27EF	514
Miscellaneous Mathematical Symbols-B: U+2980–U+29FF	514
Miscellaneous Symbols and Arrows: U+2B00–U+2B7F	515
Arrows: U+2190–U+21FF	515
Supplemental Arrows	515
Standardized Variants of Mathematical Symbols	516
15.6 Invisible Mathematical Operators	516
15.7 Technical Symbols	517
Control Pictures: U+2400–U+243F	517
Miscellaneous Technical: U+2300–U+23FF	517
Optical Character Recognition: U+2440–U+245F	520
15.8 Geometrical Symbols	520
Box Drawing and Block Elements	520
Geometric Shapes: U+25A0–U+25FF	521
15.9 Miscellaneous Symbols	522
Miscellaneous Symbols: U+2600–U+26FF	523
Miscellaneous Symbols and Pictographs: U+1F300–U+1F5FF	523
Emoticons: U+1F600–U+1F64F	525
Transport and Map Symbols: U+1F680–U+1F6FF	525
Dingbats: U+2700–U+27BF	526
Alchemical Symbols: U+1F700–U+1F77F	527
Mahjong Tiles: U+1F000–U+1F02F	527
Domino Tiles: U+1F030–U+1F09F	527
Playing Cards: U+1F0A0–U+1F0FF	528
Yijing Hexagram Symbols: U+4DC0–U+4DFF	529
Tai Xuan Jing Symbols: U+1D300–U+1D356	529
Ancient Symbols: U+10190–U+101CF	530
Phaistos Disc Symbols: U+101D0–U+101FF	530

15.10 Enclosed and Square	531
Enclosed Alphanumerics: U+2460–U+24FF	533
Enclosed CJK Letters and Months: U+3200–U+32FF	533
CJK Compatibility: U+3300–U+33FF	533
Enclosed Alphanumeric Supplement: U+1F100–U+1F1FF	534
Enclosed Ideographic Supplement: U+1F200–U+1F2FF	534
15.11 Braille	534
15.12 Western Musical Symbols	536
15.13 Byzantine Musical Symbols	540
15.14 Ancient Greek Musical Notation	540
16 Special Areas and Format Characters	543
16.1 Control Codes	544
Representing Control Sequences	544
Specification of Control Code Semantics	545
16.2 Layout Controls	545
Line and Word Breaking	546
Cursive Connection and Ligatures	548
Combining Grapheme Joiner	551
Bidirectional Ordering Controls	553
Stateful Format Controls	553
16.3 Deprecated Format Characters	554
16.4 Variation Selectors	556
16.5 Private-Use Characters	557
Private Use Area: U+E000–U+F8FF	558
Supplementary Private Use Areas	559
16.6 Surrogates Area	559
16.7 Noncharacters	560
16.8 Specials	561
Byte Order Mark (BOM): U+FEFF	561
Specials: U+FFF0–U+FFF8	563
Annotation Characters: U+FFF9–U+FFFB	563
Replacement Characters: U+FFFC–U+FFFD	564
16.9 Deprecated Tag Characters	565
Deprecated Tag Characters: U+E0000–U+E007F	565
Syntax for Embedding Tags	565
Working with Language Tags	567
Unicode Conformance Issues	568
Formal Tag Syntax	569
17 About the Code Charts	571
17.1 Character Names List	571
Images in the Code Charts and Character Lists	572
Special Characters and Code Points	573
Character Names	574
Informative Aliases	574
Normative Aliases	575
Cross References	575
Information About Languages	575
Case Mappings	575
Decompositions	576
Subheads	577

17.2	CJK Unified and Compatibility Ideographs	577
	CJK Unified Ideographs	577
	Compatibility Ideographs	579
17.3	Hangul Syllables	579
A	Notational Conventions	581
	Code Points	581
	Character Names	581
	Character Blocks	581
	Sequences	582
	Rendering	582
	Properties and Property Values	582
	Miscellaneous	582
	Extended BNF	583
	Operators	584
B	Unicode Publications and Resources	585
B.1	The Unicode Consortium	585
	The Unicode Technical Committee	585
	Other Activities	586
B.2	Unicode Publications	586
B.3	Unicode Technical Standards	586
	UTS #6: A Standard Compression Scheme for Unicode	586
	UTS #10: Unicode Collation Algorithm	587
	UTS #18: Unicode Regular Expressions	587
	UTS #22: Character Mapping Markup Language (CharMapML)	587
	UTS #35: Unicode Locale Data Markup Language (LDML)	587
	UTS #37: Unicode Ideographic Variation Database	587
	UTS #39: Unicode Security Mechanisms	587
B.4	Unicode Technical Reports	587
	UTR #16: UTF-EBCDIC	587
	UTR #17: Unicode Character Encoding Model	588
	UTR #20: Unicode in XML and Other Markup Languages	588
	UTR #23: The Unicode Character Property Model	588
	UTR #25: Unicode Support for Mathematics	588
	UTR #26: Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8) ..	588
	UTR #33: Unicode Conformance Model	588
	UTR #36: Unicode Security Considerations	588
	UTR #45: U-Source Ideographs	589
B.5	Unicode Technical Notes	589
B.6	Other Unicode Online Resources	589
	Unicode Online Resources	589
	How to Contact the Unicode Consortium	591
C	Relationship to ISO/IEC 10646	593
C.1	History	593
C.2	Encoding Forms in ISO/IEC 10646	597
	Zero Extending	597
C.3	UTF-8 and UTF-16	598
	UTF-8	598
	UTF-16	598
C.4	Synchronization of the Standards	598

C.5	Identification of Features for the Unicode Standard	598
C.6	Character Names	599
C.7	Character Functional Specifications	599
D	Changes from Previous Versions	601
D.1	Versions of the Unicode Standard	601
D.2	Clause and Definition Updates	603
E	Han Unification History	605
E.1	Development of the URO	605
E.2	Ideographic Rapporteur Group	606
F	Documentation of CJK Strokes	609
R	References	615
R.1	Source Standards and Specifications	615
R.2	Source Dictionaries for Han Unification	622
R.3	Other Sources for the Unicode Standard	622
R.4	Selected Resources: Technical	638
R.5	Selected Resources: Other	640
I	General Index	645

Figures

Figure 1-1.	Wide ASCII	2
Figure 1-2.	Unicode Compared to the 2022 Framework.	4
Figure 2-1.	Text Elements and Characters	9
Figure 2-2.	Characters Versus Glyphs	12
Figure 2-3.	Unicode Character Code to Rendered Glyphs	13
Figure 2-4.	Bidirectional Ordering	15
Figure 2-5.	Writing Direction and Numbers	16
Figure 2-6.	Typeface Variation for the Bone Character.	17
Figure 2-7.	Dynamic Composition	18
Figure 2-8.	Abstract and Encoded Characters	22
Figure 2-9.	Overlap in Legacy Mixed-Width Encodings	25
Figure 2-10.	Boundaries and Interpretation	25
Figure 2-11.	Unicode Encoding Forms	26
Figure 2-12.	Unicode Encoding Schemes	31
Figure 2-13.	Unicode Allocation	36
Figure 2-14.	Allocation on the BMP	37
Figure 2-15.	Allocation on Plane 1.	39
Figure 2-16.	Writing Directions.	40
Figure 2-17.	Combining Enclosing Marks for Symbols.	42
Figure 2-18.	Sequence of Base Characters and Diacritics	42
Figure 2-19.	Reordered Indic Vowel Signs	43
Figure 2-20.	Properties and Combining Character Sequences	43
Figure 2-21.	Stacking Sequences	43
Figure 2-22.	Ligated Multiple Base Characters.	45
Figure 2-23.	Equivalent Sequences	47
Figure 2-24.	Canonical Ordering.	47
Figure 2-25.	Types of Decomposables.	49
Figure 3-1.	Enclosing Marks.	85
Figure 4-1.	Positions of Common Combining Marks	127
Figure 5-1.	Two-Stage Tables	147
Figure 5-2.	Normalization	152
Figure 5-3.	Consistent Character Boundaries.	158
Figure 5-4.	Dead Keys Versus Handwriting Sequence.	161
Figure 5-5.	Truncating Grapheme Clusters	161
Figure 5-6.	Inside-Out Rule	162
Figure 5-7.	Fallback Rendering	163
Figure 5-8.	Bidirectional Placement	163
Figure 5-9.	Justification.	164
Figure 5-10.	Positioning with Ligatures	165
Figure 5-11.	Positioning with Contextual Forms	166
Figure 5-12.	Positioning with Enhanced Kerning	167
Figure 5-13.	Sublinear Searching	169
Figure 5-14.	Uppercase Mapping for Turkish I	173
Figure 5-15.	Lowercase Mapping for Turkish I	174
Figure 5-16.	Casing of German Sharp S	174
Figure 6-1.	Overriding Inherent Vowels	189
Figure 6-2.	Forms of CJK Punctuation	192
Figure 6-3.	European Quotation Marks	198

Figure 6-4.	Asian Quotation Marks	199
Figure 6-5.	Examples of Ancient Greek Editorial Marks	205
Figure 6-6.	Use of Greek Paragraphos	206
Figure 6-7.	CJK Parentheses	207
Figure 7-1.	Alternative Glyphs in Latin	213
Figure 7-2.	Diacritics on <i>i</i> and <i>j</i>	214
Figure 7-3.	Vietnamese Letters and Tone Marks	214
Figure 7-4.	Variations in Greek Capital Letter Upsilon	224
Figure 7-5.	Coptic Numerals	229
Figure 7-6.	Georgian Scripts and Casing	234
Figure 7-7.	Tone Letters	237
Figure 7-8.	Double Diacritics	240
Figure 7-9.	Positioning of Double Diacritics	240
Figure 7-10.	Use of CGJ with Double Diacritics	241
Figure 7-11.	Interaction of Combining Marks with Ligatures	241
Figure 7-12.	Use of Vertical Line Overlay for Negation	243
Figure 7-13.	Double Diacritics and Half Marks	244
Figure 8-1.	Directionality and Cursive Connection	250
Figure 8-2.	Using a Joiner	252
Figure 8-3.	Using a Non-joiner	252
Figure 8-4.	Combinations of Joiners and Non-joiners	252
Figure 8-5.	Placement of Harakat	253
Figure 8-6.	Arabic Year Sign	255
Figure 8-7.	Syriac Abbreviation	268
Figure 8-8.	Use of SAM	268
Figure 9-1.	Dead Consonants in Devanagari	281
Figure 9-2.	Conjunct Formations in Devanagari	282
Figure 9-3.	Preventing Conjunct Forms in Devanagari	282
Figure 9-4.	Half-Consonants in Devanagari	283
Figure 9-5.	Independent Half-Forms in Devanagari	283
Figure 9-6.	Half-Consonants in Oriya	283
Figure 9-7.	Consonant Forms in Devanagari and Oriya	284
Figure 9-8.	Rendering Order in Devanagari	288
Figure 9-9.	Marathi Allographs	291
Figure 9-10.	Use of Apostrophe in Bodo, Dogri and Maithili	292
Figure 9-11.	Use of Avagraha in Dogri	292
Figure 9-12.	Requesting Bengali Consonant-Vowel Ligature	297
Figure 9-13.	Blocking Bengali Consonant-Vowel Ligature	297
Figure 9-14.	Bengali Syllable tta	298
Figure 9-15.	Kssa Ligature in Tamil	307
Figure 9-16.	Tamil Two-Part Vowels	308
Figure 9-17.	Vowel Reordering Around a Tamil Conjunct	308
Figure 9-18.	Tamil Ligatures with <i>i</i>	309
Figure 9-19.	Spacing Forms of Tamil <i>u</i>	309
Figure 9-20.	Tamil Ligatures with <i>ra</i>	310
Figure 9-21.	Traditional Tamil Ligatures with <i>aa</i>	310
Figure 9-22.	Traditional Tamil Ligatures with <i>o</i>	310
Figure 9-23.	Traditional Tamil Ligatures with <i>ai</i>	311
Figure 9-24.	Vowel <i>ai</i> in Modern Tamil	311
Figure 10-1.	Tibetan Syllable Structure	327
Figure 10-2.	Justifying Tibetan Tseks	334
Figure 10-3.	Phags-pa Syllable Om	338
Figure 10-4.	Phags-pa Reversed Shaping	341
Figure 10-5.	Geographical Extent of the Kharoshthi Script	355

Figure 10-6.	Kharoshthi Number 1996	356
Figure 10-7.	Kharoshthi Rendering Example	356
Figure 10-8.	Consonant Ligatures in Brahmi	360
Figure 11-1.	Common Ligatures in Khmer	380
Figure 11-2.	Common Multiple Forms in Khmer	380
Figure 11-3.	Examples of Syllabic Order in Khmer	382
Figure 11-4.	Ligation in <i>Muul</i> Style in Khmer	382
Figure 11-5.	Buginese Ligature.	394
Figure 11-6.	Writing dharma in Balinese	397
Figure 11-7.	Representation of Javanese Two-Part Vowels.	400
Figure 12-1.	Han Spelling	412
Figure 12-2.	Semantic Context for Han Characters.	413
Figure 12-3.	Three-Dimensional Conceptual Model.	415
Figure 12-4.	CJK Source Separation	415
Figure 12-5.	Not Cognates, Not Unified.	416
Figure 12-6.	Ideographic Component Structure	417
Figure 12-7.	The Most Superior Node of an Ideographic Component	417
Figure 12-8.	Using the Ideographic Description Characters.	425
Figure 12-9.	Japanese Historic Kana for e and ye.	429
Figure 13-1.	Mongolian Glyph Convergence	443
Figure 13-2.	Mongolian Consonant Ligation	443
Figure 13-3.	Mongolian Positional Forms	443
Figure 13-4.	Mongolian Free Variation Selector	444
Figure 13-5.	Mongolian Gender Forms.	446
Figure 13-6.	Mongolian Vowel Separator	446
Figure 13-7.	Tifinagh Contextual Shaping	449
Figure 13-8.	Tifinagh Consonant Joiner and Bi-consonants.	450
Figure 13-9.	Examples of N’Ko Ordinals	452
Figure 13-10.	Short Words Equivalent to Deseret Letter Names	460
Figure 14-1.	Distribution of Old Italic.	469
Figure 14-2.	Interpretion of Hieroglyphic Markup	489
Figure 15-1.	Alternative Glyphs for Dollar Sign	494
Figure 15-2.	Alternative Glyphs for Numero Sign	496
Figure 15-3.	Wide Mathematical Accents	498
Figure 15-4.	Style Variants and Semantic Distinctions in Mathematics	499
Figure 15-5.	Easily Confused Shapes for Mathematical Glyphs	500
Figure 15-6.	CJK Ideographic Numbers	503
Figure 15-7.	Regular and Old Style Digits.	505
Figure 15-8.	Alternate Forms of Vulgar Fractions	509
Figure 15-9.	Usage of Crops and Quine Corners	518
Figure 15-10.	Usage of the Decimal Exponent Symbol	520
Figure 15-11.	Examples of Specialized Music Layout	538
Figure 15-12.	Precomposed Note Characters.	538
Figure 15-13.	Alternative Noteheads	539
Figure 15-14.	Augmentation Dots and Articulation Symbols.	539
Figure 16-1.	Prevention of Joining.	549
Figure 16-2.	Exhibition of Joining Glyphs in Isolation	550
Figure 16-3.	Effect of Intervening Joiners	550
Figure 16-4.	Annotation Characters	563
Figure 16-5.	Tag Characters	566
Figure 17-1.	CJK Chart Format for the Main CJK Block.	578
Figure 17-2.	CJK Chart Format for CJK Extension A	578
Figure 17-3.	CJK Chart Format for CJK Extension B	579
Figure 17-4.	CJK Chart Format for Compatibility Ideographs.	579

Figure A-1. Example of Rendering..... 582

Tables

Table 2-1.	The 10 Unicode Design Principles	11
Table 2-2.	User-Perceived Characters with Multiple Code Points	12
Table 2-3.	Types of Code Points	23
Table 2-4.	The Seven Unicode Encoding Schemes	30
Table 2-5.	Interaction of Combining Characters	44
Table 2-6.	Nondefault Stacking	45
Table 3-1.	Named Unicode Algorithms	69
Table 3-2.	Normative Character Properties	74
Table 3-3.	Informative Character Properties	75
Table 3-4.	Examples of Unicode Encoding Forms	93
Table 3-5.	UTF-16 Bit Distribution	94
Table 3-6.	UTF-8 Bit Distribution	94
Table 3-7.	Well-Formed UTF-8 Byte Sequences	95
Table 3-8.	Use of U+FFFD in UTF-8 Conversion	97
Table 3-9.	Summary of UTF-16BE, UTF-16LE, and UTF-16	99
Table 3-10.	Summary of UTF-32BE, UTF-32LE, and UTF-32	100
Table 3-11.	Combining Marks and Starter Status	103
Table 3-12.	Reorderable Pairs	104
Table 3-13.	Hangul Characters Used in Examples	109
Table 3-14.	Context Specification for Casing	116
Table 3-15.	Case Detection Examples	119
Table 4-1.	Relationship of Casing Definitions	125
Table 4-2.	Case Function Values for Strings	125
Table 4-3.	Sources for Case Mapping Information	126
Table 4-4.	Class Zero Combining Marks—Reordrant	128
Table 4-5.	Thai, Lao, and Tai Viet Logical Order Exceptions	128
Table 4-6.	Class Zero Combining Marks—Split	129
Table 4-7.	Class Zero Combining Marks—Subjoined	130
Table 4-8.	Class Zero Combining Marks—Strikethrough	130
Table 4-9.	General Category	131
Table 4-10.	Primary Numeric Ideographs	134
Table 4-11.	Ideographs Used as Accounting Numbers	134
Table 4-12.	Construction of Code Point Labels	138
Table 4-13.	Unusual Properties	141
Table 5-1.	Hex Values for Acronyms	154
Table 5-2.	NLF Platform Correlations	154
Table 5-3.	Typing Order Differing from Canonical Order	164
Table 5-4.	Permuting Combining Class Weights	165
Table 5-5.	Casing and Normalization in Strings	177
Table 6-1.	Typology of Scripts in the Unicode Standard	191
Table 6-2.	Unicode Space Characters	194
Table 6-3.	Unicode Dash Characters	196
Table 6-4.	East Asian Quotation Marks	199
Table 6-5.	Opening and Closing Forms	199
Table 6-6.	Names for the @	203
Table 6-7.	Unicode Danda Characters	206
Table 7-1.	Nonspacing Marks Used with Greek	222
Table 7-2.	Greek Spacing and Nonspacing Pairs	226

Table 8-1.	Arabic Digit Names	254
Table 8-2.	Glyph Variation in Eastern Arabic-Indic Digits	254
Table 8-3.	Primary Arabic Joining Types	257
Table 8-4.	Derived Arabic Joining Types	257
Table 8-5.	Arabic Glyph Types	257
Table 8-6.	Arabic Obligatory Ligature Joining Groups	259
Table 8-7.	Arabic Ligature Notation	259
Table 8-8.	Dual-Joining Arabic Characters	260
Table 8-9.	Right-Joining Arabic Characters	261
Table 8-10.	Forms of the Arabic Letter yeh	262
Table 8-11.	Arabic Letters With Hamza Above	264
Table 8-12.	Miscellaneous Syriac Diacritic Use	270
Table 8-13.	Syriac Final Alaph Glyph Types	270
Table 8-14.	Dual-Joining Syriac Characters	271
Table 8-15.	Right-Joining Syriac Characters	272
Table 8-16.	Syriac Alaph Glyph Forms	272
Table 8-17.	Syriac Ligatures	272
Table 8-18.	Samaritan Performative Punctuation Marks	274
Table 8-19.	Thaana Glyph Placement	275
Table 9-1.	Devanagari Vowel Letters	281
Table 9-2.	Sample Devanagari Half-Forms	289
Table 9-3.	Sample Devanagari Ligatures	290
Table 9-4.	Sample Devanagari Half-Ligature Forms	291
Table 9-5.	Devanagari Vowels Used in Bihari Languages	293
Table 9-6.	Prishthamatra Orthography	293
Table 9-7.	Bengali Vowel Letters	296
Table 9-8.	Bengali Consonant-Vowel Combinations	297
Table 9-9.	Use of Apostrophe in Bangla	299
Table 9-10.	Gurmukhi Vowel Letters	301
Table 9-11.	Gurmukhi Conjuncts	302
Table 9-12.	Additional Pairin and Addha Forms in Gurmukhi	302
Table 9-13.	Use of Joiners in Gurmukhi	303
Table 9-14.	Gujarati Vowel Letters	303
Table 9-15.	Gujarati Conjuncts	304
Table 9-16.	Oriya Vowel Letters	304
Table 9-17.	Oriya Conjuncts	305
Table 9-18.	Oriya Vowel Placement	305
Table 9-19.	Tamil Vowel Reordering	307
Table 9-20.	Tamil Vowel Splitting and Reordering	308
Table 9-21.	Tamil Ligatures with u	309
Table 9-22.	Tamil Vowels, Consonants, and Syllables	312
Table 9-23.	Telugu Vowel Letters	313
Table 9-24.	Rendering of Telugu na + virama	314
Table 9-25.	Kannada Vowel Letters	316
Table 9-26.	Malayalam Vowel Letters	318
Table 9-27.	Malayalam Orthographic Reform	318
Table 9-28.	Malayalam Conjuncts	319
Table 9-29.	Candrakala Examples	319
Table 9-30.	Atomic Encoding of Malayalam Chillus	320
Table 9-31.	Malayalam /rr/ and /tt/	320
Table 9-32.	Malayalam /nr/ and /nt/	321
Table 10-1.	Sinhala Vowel Letters	325
Table 10-2.	Lepcha Syllabic Structure	336
Table 10-3.	Phags-pa Positional Forms of I, U, E, and O	340

Table 10-4.	Contextual Glyph Mirroring in Phags-pa	340
Table 10-5.	Phags-pa Standardized Variants	341
Table 10-6.	Positions of Limbu Combining Characters	344
Table 10-7.	Takri Vowel Letters	350
Table 10-8.	Kharoshthi Vowel Signs	357
Table 10-9.	Kharoshthi Vowel Modifiers	358
Table 10-10.	Kharoshthi Consonant Modifiers	358
Table 10-11.	Examples of Kharoshthi Virama	359
Table 10-12.	Brahmi Vowel Letters	359
Table 10-13.	Brahmi Positional Digits	361
Table 11-1.	Glyph Positions in Thai Syllables	365
Table 11-2.	Glyph Positions in Lao Syllables	367
Table 11-3.	Myanmar Syllabic Structure	371
Table 11-4.	Khamti Shan Tone Marks	372
Table 11-5.	Independent Khmer Vowel Characters	374
Table 11-6.	Two Registers of Khmer Consonants	376
Table 11-7.	Khmer Subscript Consonant Signs	377
Table 11-8.	Khmer Composite Dependent Vowel Signs with Nikahit	378
Table 11-9.	Khmer Subscript Independent Vowel Signs	379
Table 11-10.	Tai Le Tone Marks	383
Table 11-11.	Myanmar Digits	384
Table 11-12.	New Tai Lue Vowel Placement	385
Table 11-13.	New Tai Lue Registers and Tones	385
Table 11-14.	Tai Viet Symbols and Punctuation	389
Table 11-15.	Cham Syllabic Structure	391
Table 11-16.	Hanunóo and Buhid Vowel Sign Combinations	393
Table 11-17.	Balinese Base Consonants and Conjunct Forms	395
Table 11-18.	Sasak Extensions for Balinese	396
Table 11-19.	Balinese Consonant Clusters with u and u:	397
Table 11-20.	Sundanese Syllabic Structure	404
Table 12-1.	Sources for Unified Han	407
Table 12-2.	Blocks Containing Han Ideographs	409
Table 12-3.	Small Extensions to the URO	410
Table 12-4.	Common Han Characters	411
Table 12-5.	Source Encoding for Sword Variants	416
Table 12-6.	Ideographs Not Unified	417
Table 12-7.	Ideographs Unified	418
Table 12-8.	Han Ideograph Arrangement	418
Table 12-9.	Mandarin Tone Marks	426
Table 12-10.	Minnan and Hakka Tone Marks	427
Table 12-11.	Separating Jamo Characters	431
Table 12-12.	Line-Based Placement of Jungseong	432
Table 13-1.	Labialized Forms in Ethiopic -WAA	438
Table 13-2.	Labialized Forms in Ethiopic -WE	439
Table 13-3.	N’Ko Tone Diacritics on Vowels	451
Table 13-4.	Other N’Ko Diacritic Usage	452
Table 13-5.	N’Ko Letter Shaping	453
Table 13-6.	IPA Transcription of Deseret	460
Table 13-7.	Lisu Tone Letters	462
Table 13-8.	Punctuation Adopted in Lisu Orthography	463
Table 14-1.	Similar Characters in Linear B and Cypriot	474
Table 14-2.	Old South Arabian Numeric Characters	476
Table 14-3.	Number Formation in Old South Arabian	477
Table 14-4.	Number Formation in Aramaic	479

Table 14-5.	Dual-Joining Mandaic Characters	480
Table 14-6.	Right-Joining Mandaic Characters	481
Table 14-7.	Inscriptional Parthian Shaping Behavior	482
Table 14-8.	Avestan Shaping Behavior	483
Table 14-9.	Cuneiform Script Usage	486
Table 14-10.	Hieroglyphic Character Sequence	489
Table 15-1.	Currency Symbols Encoded in Other Blocks	495
Table 15-2.	Mathematical Alphanumeric Symbols	499
Table 15-3.	Script-Specific Decimal Digits	502
Table 15-4.	Compatibility Digits	504
Table 15-5.	Use of Mathematical Symbol Pieces	519
Table 15-6.	Japanese Era Names	533
Table 15-7.	Examples of Ornamentation	539
Table 15-8.	Representation of Ancient Greek Vocal and Instrumental Notation	541
Table 16-1.	Control Codes Specified in the Unicode Standard	545
Table 16-2.	Letter Spacing	547
Table 16-3.	Bidirectional Ordering Controls	553
Table 16-4.	Paired Stateful Controls	554
Table 16-5.	Paired Stateful Controls (Deprecated)	554
Table 16-6.	Unicode Encoding Scheme Signatures	562
Table 16-7.	U+FEFF Signature in Other Charsets	563
Table 17-1.	IRG Sources	577
Table A-1.	Extended BNF	583
Table A-2.	Character Class Examples	584
Table A-3.	Operators	584
Table C-1.	Timeline	594
Table C-2.	Zero Extending	598
Table D-1.	Versions of Unicode and ISO/IEC 10646-1	601
Table D-2.	Allocation of Code Points by Type	602
Table D-3.	Allocation of Code Points by Type (Early Versions)	602
Table D-4.	Version 5.1 Clause and Definition Updates	603
Table D-5.	Version 5.2 Clause and Definition Updates	603
Table D-6.	Version 6.0 Clause and Definition Updates	604
Table D-7.	Version 6.1 Clause and Definition Updates	604
Table F-1.	CJK Strokes	610

Preface

This is *The Unicode Standard, Version 6.1*. It supersedes all earlier versions of the Unicode Standard. Version 6.1 is published solely in online format.

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 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.

What’s New?

Key new features that have been defined and documented since the publication of *The Unicode Standard, Version 6.0* include:

- additional characters for languages of China, other Asian countries, and Africa
- new math characters to support educational needs in the Arabic-speaking world
- labels for properties that aid implementation
- consolidation of Hangul algorithms
- support of new display styles for many *emoji* characters
- improved line-breaking behavior of Hebrew and Japanese text
- improved segmentation behavior for Thai, Lao, and similar languages
- more fully specified mappings between Simplified and Traditional Chinese characters

Support for Languages and Symbol Sets. 732 new characters were added in the Unicode Standard, Version 6.1. New Arabic and Latin characters were added to support languages of Africa, including those used in Cameroon, Chad, Guinea, Nigeria, and Senegal, as well as for languages of Laos, Myanmar, and the Philippines. The characters used in Chad are required by its Department of Education. Support for other regional languages and scripts added in Version 6.1 includes:

- Miao, used in China, Vietnam, Laos, and Thailand
- Chakma, used in India and Bangladesh
- Meetei Mayek, used in India

- Sundanese, used in Indonesia

Symbol additions include new math symbols, *emoji* symbols, Koranic annotation signs, and the Armenian Dram currency symbol. Version 6.1 also added other scripts: Sharada, Sora Sompeng, Takri, Meroitic Cursive, and Meroitic Hieroglyphs.

Detailed Change Information. See *Appendix D, Changes from Previous Versions* and <http://www.unicode.org/versions/Unicode6.1.0/> for detailed information about the changes from the previous versions of the standard, including character counts, conformance clause and definition updates, and significant changes to the Unicode Character Database and Unicode Standard Annexes.

Organization of This Standard

This core specification, together with the Unicode code charts, the Unicode Character Database, and the Unicode Standard Annexes, defines Version 6.1 of the Unicode Standard. The core specification contains the general principles, requirements for conformance, and guidelines for implementers. The character code charts and names are also available online.

This core specification, together with the Unicode code charts, the Unicode Character Database, and the Unicode Standard Annexes, defines Version 6.1 of the Unicode Standard. The core specification contains the general principles, requirements for conformance, and guidelines for implementers. The character code charts and names are also available online.

Concepts, Architecture, Conformance, and Guidelines. The first five chapters of Version 6.1 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 16* 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 17* 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 abstracts of Unicode Technical Reports and lists other important Unicode resources.

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

Appendix D lists the changes to the Unicode Standard since Version 5.0.

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+C130..U+31EF).

References and Index. The appendices are followed by a bibliography and an index to the text of this core specification.

Glossary and Character Index. A glossary of Unicode terms and the Unicode Character Name Index may be found at:

<http://www.unicode.org/glossary/>

<http://www.unicode.org/charts/charindex.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:

<http://www.unicode.org/reports/>

The following is a 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,” discusses the Script property specified in the Unicode Character Database.

Unicode Standard Annex #29, “Unicode Text Segmentation,” describes algorithms for determining default boundaries between certain signifi-

cant text elements: grapheme clusters (“user-perceived characters”), words, and sentences.

Unicode Standard Annex #31, “Unicode Identifier and Pattern 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.

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:

<http://www.unicode.org/ucd/>

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

<http://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 6.1 code charts may be found at:

<http://www.unicode.org/charts/PDF/Unicode-6.1/>

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

<http://www.unicode.org/charts/>

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

<http://www.unicode.org/charts/unihanrsindex.html>

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.

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

<http://www.unicode.org/reports/>

See *Appendix B, Unicode Publications and Resources*, for a summary overview of important Unicode Technical Standards and Unicode Technical 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:

<http://www.unicode.org/reporting.html>

A list of known errata is maintained at:

<http://www.unicode.org/errata/>

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

Acknowledgements

The Unicode Standard, Version 6.1 is the result of the dedication and contributions of many people over several 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:

<http://www.unicode.org/acknowledgements/>

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*).

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, Version 6.1, 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.

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
l	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

1.1 Coverage

The Unicode Standard, Version 6.1, contains 110,116 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 74,616 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.

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.

For a complete list of ISO and national standards used as sources, see *References*.

The Unicode Standard is fully conformant with the International Standard ISO/IEC 10646:2011, *Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Architecture and Basic Multilingual Plane, Supplementary Planes*, known as the Universal Character Set (UCS). For more information, see *Appendix C, Relationship to ISO/IEC 10646*.

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.

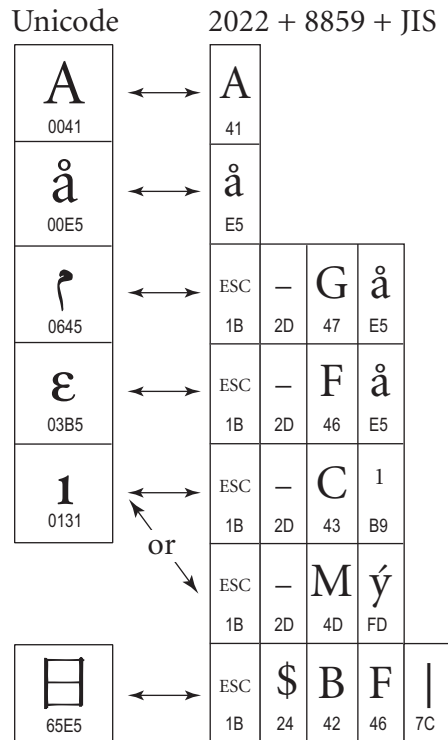
The designers of the Unicode Standard envisioned a uniform method of character identification that would be more efficient and flexible than previous encoding systems. The new system would satisfy the needs of technical and multilingual computing and would encode a broad range of characters for all purposes,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

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.

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 definition 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—10 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 *Chapter 3, Conformance*.

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.

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

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.

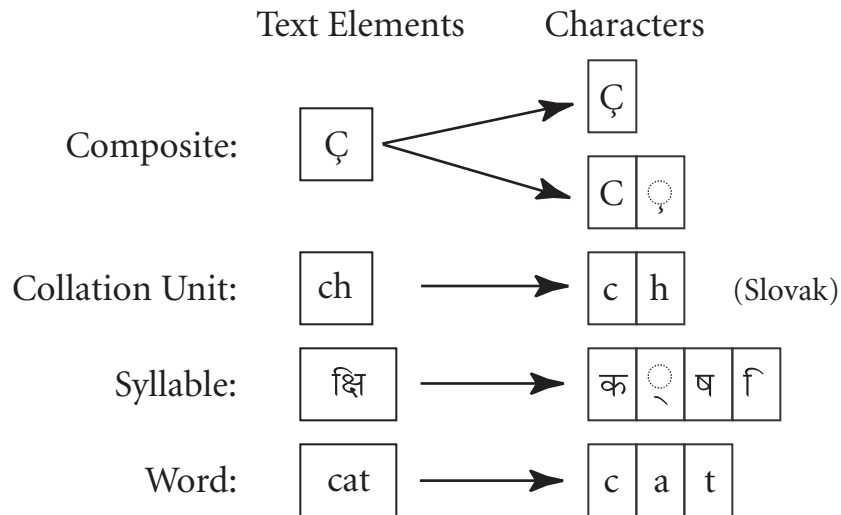
For example, in traditional German orthography, the letter combination “ck” is a text element for the process of hyphenation (where it appears as “k-k”), but not for the process of sorting. In Spanish, the combination “ll” may be a text element for the traditional process of sorting (where it is sorted 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. Unicode Standard Annex #29, “Unicode Text Segmentation,” provides more details regarding the specifications of boundaries.

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*.

Figure 2-1. Text Elements and Characters



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 8.2, Arabic*, and *Section 9.1, 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, rather than unconsciously.

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.

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.

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.

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.

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.

All Unicode encoding forms 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.

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	U+0041 LATIN CAPITAL LETTER A
a a a a a a a a	U+0061 LATIN SMALL LETTER A
П п 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

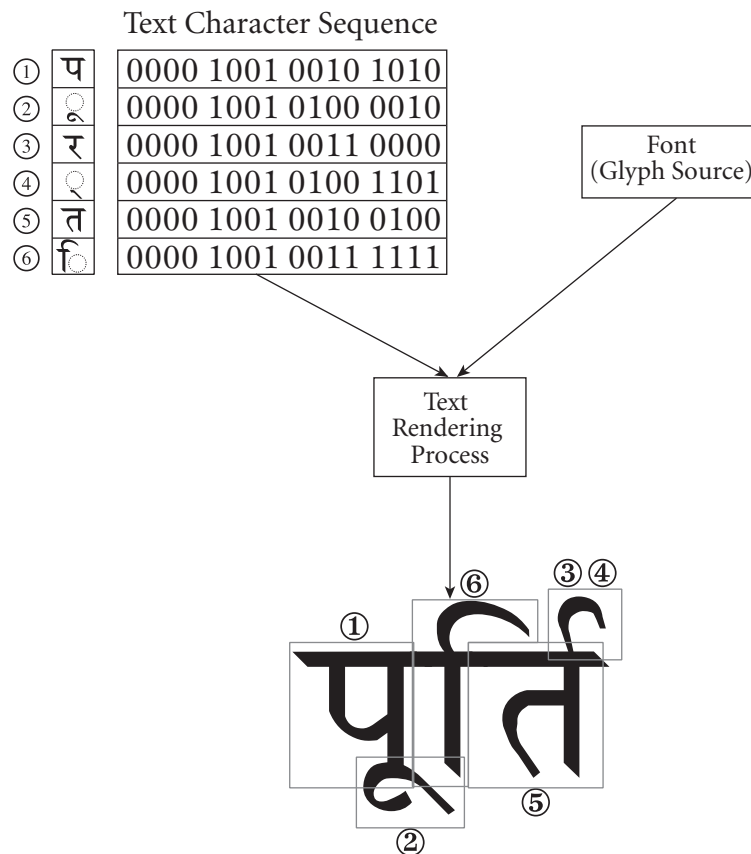
Character	Code Points	Linguistic Usage
ch	0063 0068	Slovak, traditional Spanish
t ^h	0074 02B0	Native American languages
Ḫ	0078 0323	
Ḫ̂	019B 0313	
ą	00E1 0328	Lithuanian
į	0069 0307 0301	
ト	30C8 309A	Ainu (in kana transcription)

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 *Figure 2-3*.

Figure 2-3. Unicode Character Code to Rendered Glyphs



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.

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 *Section B.6, Other Unicode Online Resources*.

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.

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.

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

G	i	d	i	_	s	a	i	d	,	_	“	א	ם	_	א	י	_	א	נ	י	_	ל	י	_	ב	י	_	ל	י	”	.
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

G	i	d	i	s	a	i	d	,	“	א	ם	א	י	א	נ	י	ל	י	ב	י	ל	י	”	.
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

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 90 degrees to follow the layout direction of the lines, as shown in Example 6.

Figure 2-5. Writing Direction and Numbers

- | | |
|---|--|
| <p>① آ ٥٦ ٧٨٩٠ ١٢٣٤ ٥٦٧٨٩٠</p> <p>② .1123 גאראה לעמוד</p> <p>③ .راجع صفحة ١١٢٣ من فضلك.</p> <p>④ Please see page 1123.</p> <p>⑤ 1123ページをみてください。</p> | <p>⑥ み 1
て 1
く 2
だ 2
さ 3
い 3
。 3
を 3
を 3</p> |
|---|--|

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 “x̄” 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 “x̄” 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 and Thai Viet 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.

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 12.1, 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 ETHIOPIAN FULL STOP, U+060F ARABIC SIGN MISRA, and U+0592 HEBREW ACCENT SEGOL. However, some punctuation or diacritic 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+0057 “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 12.1, 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



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.

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

$$\begin{array}{c} \text{A} + \text{¨} \rightarrow \text{Ä} \\ \text{0041} \quad \text{0308} \end{array}$$

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 and Normalization*).

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, spell-checkers 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:

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

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 8.2, 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 Unicode Technical Report #20, “Unicode and 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.

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.

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 Versus 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.

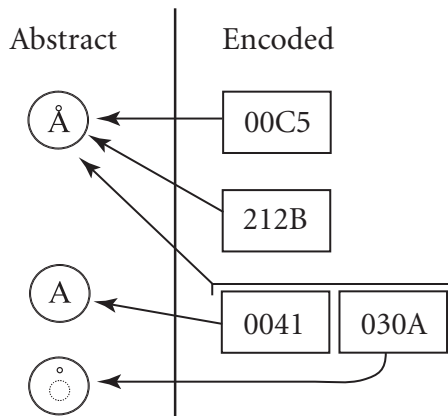
The range of integers used to code the abstract characters is called the *codespace*. A particular integer in this set is called a *code point*. When an abstract character is mapped or *assigned* to a particular code point in the codespace, it is then referred to as an *encoded character*.

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.

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	Not assigned to abstract character	
Surrogate	Permanently reserved for UTF-16; restricted interchange	Cs		
Noncharacter	Permanently reserved for internal usage; restricted interchange	Cn		
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 16.1, 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 FFFE₁₆ or FFFF₁₆—that is, U+FFFE, U+FFFF, U+1FFFE, U+1FFFF, ... U+10FFFE, U+10FFFF. (See *Section 16.7, 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 16.5, 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 16.6, 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 should never be interchanged. They do, however, 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 5.2 system and a second person is using a Unicode 3.2 system. The first person sends the second person a document containing some code points newly assigned in Unicode 5.2; these code points were unassigned in Unicode 3.2. 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 16-1* in *Section 16.1, 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.

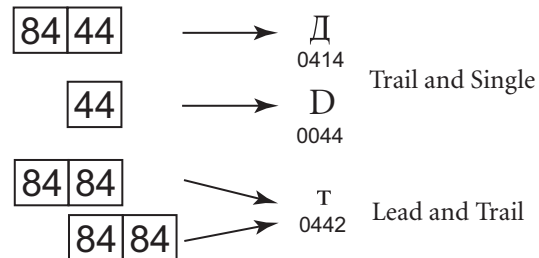
Actual implementations in computer systems represent integers in specific *code units* of particular size—usually 8-bit (= byte), 16-bit, or 32-bit. In the Unicode character encoding model, precisely defined *encoding forms* specify how each integer (code point) for a Unicode character is to be expressed 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. How-

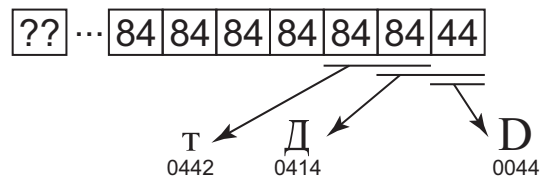
ever, single-byte values and trail-byte values can overlap. That means that when someone searches for the character “D”, for example, he or she might find it either (mistakenly) as the trail byte of a two-byte sequence or as a single, independent byte. To find out which alternative is correct, a program 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

A	Ω	語	𐄎	UTF-32
00000041	000003A9	00008A9E	00010384	
A	Ω	語	𐄎	UTF-16
0041	03A9	8A9E	D800 DF84	
A	Ω	語	𐄎	UTF-8
41	CE A9	E8 AA 9E	F0 90 8E 84	

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*.

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. This makes UTF-32 an ideal form for APIs that pass single character values.

As for all of the Unicode encoding forms, UTF-32 is restricted to representation of code points in the range $0..10FFFF_{16}$ —that is, the Unicode codespace. 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. UTF-32 is also a preferred encoding form for processing characters on most Unix platforms.

UTF-16

In the UTF-16 encoding form, 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, the range U+0000..U+FFFF. 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 formally 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.

Collation. For the purpose of sorting text, 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*.

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* on page 94 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.

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 C or other programming languages, and many existing APIs that work for typical 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.

Comparison of the Advantages of UTF-32, UTF-16, and UTF-8

On the face of it, UTF-32 would seem to be the obvious choice of Unicode encoding forms for an internal processing code because it is a fixed-width encoding form. It can be conformantly bound to the C and C++ `wchar_t`, which means that such programming languages may offer built-in support and ready-made string APIs that programmers can take advantage of. However, UTF-16 has many countervailing advantages that may lead implementers to choose it instead as an internal processing code.

While all three encoding forms need at most 4 bytes (or 32 bits) of data for each character, in practice UTF-32 in almost all cases for real data sets occupies twice the storage that UTF-16 requires. Therefore, a common strategy is to have internal string storage use UTF-16 or UTF-8 but to use UTF-32 when manipulating individual characters.

UTF-32 Versus UTF-16. On average, more than 99 percent of all UTF-16 data is expressed using single code units. This includes nearly all of the typical characters that software needs to handle with special operations on text—for example, format control characters. As a consequence, most text scanning operations do not need to unpack UTF-16 surrogate pairs at all, but rather can safely treat them as an opaque part of a character string.

For many operations, UTF-16 is as easy to handle as UTF-32, and the performance of UTF-16 as a processing code tends to be quite good. UTF-16 is the internal processing code of choice for a majority of implementations supporting Unicode. Other than for Unix platforms, UTF-16 provides the right mix of compact size with the ability to handle the occasional character outside the BMP.

UTF-32 has somewhat of an advantage when it comes to simplicity of software coding design and maintenance. Because the character handling is fixed width, UTF-32 processing does not require maintaining branches in the software to test and process the double code unit elements required for supplementary characters by UTF-16. Conversely, 32-bit indices into large tables are not particularly memory efficient. To avoid the large memory penalties of such indices, Unicode tables are often handled as multistage tables (see “Multistage Tables” in *Section 5.1, Data Structures for Character Conversion*). In such cases, the 32-bit code point values are sliced into smaller ranges to permit segmented access to the tables. This is true even in typical UTF-32 implementations.

The performance of UTF-32 as a processing code may actually be worse than the performance of UTF-16 for the same data, because the additional memory overhead means that cache limits will be exceeded more often and memory paging will occur more frequently. For systems with processor designs that impose penalties for 16-bit aligned access but have very large memories, this effect may be less noticeable.

Characters Versus Code Points. In any event, Unicode code points do *not* necessarily match user expectations for “characters.” For example, the following are not represented by a single code point: a combining character sequence such as <g, acute>; a combining jamo sequence for Korean; or the Devanagari conjunct “ksha.” Because some Unicode text processing must be aware of and handle such sequences of characters as text elements, the fixed-width encoding form advantage of UTF-32 is somewhat offset by the inherently variable-width nature of processing text elements. See Unicode Technical Standard #18, “Unicode Regular Expressions,” for an example where commonly implemented processes deal with inherently variable-width text elements owing to user expectations of the identity of a “character.”

UTF-8. UTF-8 is reasonably compact in terms of the number of bytes used. It is really only at a significant size disadvantage when used for East Asian implementations such as Chinese, Japanese, and Korean, which use Han ideographs or Hangul syllables requiring three-byte code unit sequences in UTF-8. UTF-8 is also significantly less efficient in terms of processing than the other encoding forms.

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.

All three encoding forms give the same results for binary string comparisons or string sorting when dealing only with BMP characters (in the range U+0000..U+FFFF). However, when dealing with supplementary characters (in the range U+10000..U+10FFFF), UTF-16 binary order does not match Unicode code point order. This can lead to complications when trying to interoperate with binary sorted lists—for example, between UTF-16 systems and UTF-8 or UTF-32 systems. However, for data that is sorted according to the conventions of a specific language or locale rather than using binary order, data will be ordered the same, regardless of the encoding form.

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 16.8, 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 neither required nor recommended 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 16.8, Specials*, for more information.

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	𐄀	UTF-32BE
00 00 00 41	00 00 03 A9	00 00 8A 9E	00 01 03 84	
A	Ω	語	𐄀	UTF-32LE
41 00 00 00	A9 03 00 00	9E 8A 00 00	84 03 01 00	
A	Ω	語	𐄀	UTF-16BE
00 41	03 A9	8A 9E	D8 00 DF 84	
A	Ω	語	𐄀	UTF-16LE
41 00	A9 03	9E 8A	00 D8 84 DF	
A	Ω	語	𐄀	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. (See *Section 3.9, Unicode Encoding Forms*.) 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. For more information on this topic, see Unicode Technical Standard #22, “Character Mapping Markup Language (CharMapML).”

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*.

Planes

The Unicode codespace consists of the single range of numeric values from 0 to 10FFFF₁₆, 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 3456₁₆ 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.

Supplementary Ideographic Plane. The Supplementary Ideographic Plane (SIP, or Plane 2) is intended as an additional allocation area 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, in their entirety, for private use. Those two planes contain a total of 131,068 characters to supplement the 6,400 private-use characters located in the BMP.

Allocation Areas and Character 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*, which are normatively defined, and which are used to structure the actual code charts. For a complete listing of the normative character blocks in the Unicode Standard, see `Blocks.txt` in the Unicode Character Database.

The normative status of character blocks should not, however, be taken as indicating that they define significant sets of characters. For the most part, the character 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 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 13 different blocks: Basic Latin, Latin-1 Supplement, Latin Extended-A, Latin Extended-B, Latin Extended-C, Latin Extended-D, 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 Unicode character blocks 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. The empty spaces will be filled with future script encodings on a space-available basis. The relevant character encoding committees follow an organized roadmap to

help them decide where to encode new scripts within the available space. Until the characters for a script 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 may be encoded based on the identity of neighboring blocks of characters already encoded.

Assignment of Code Points

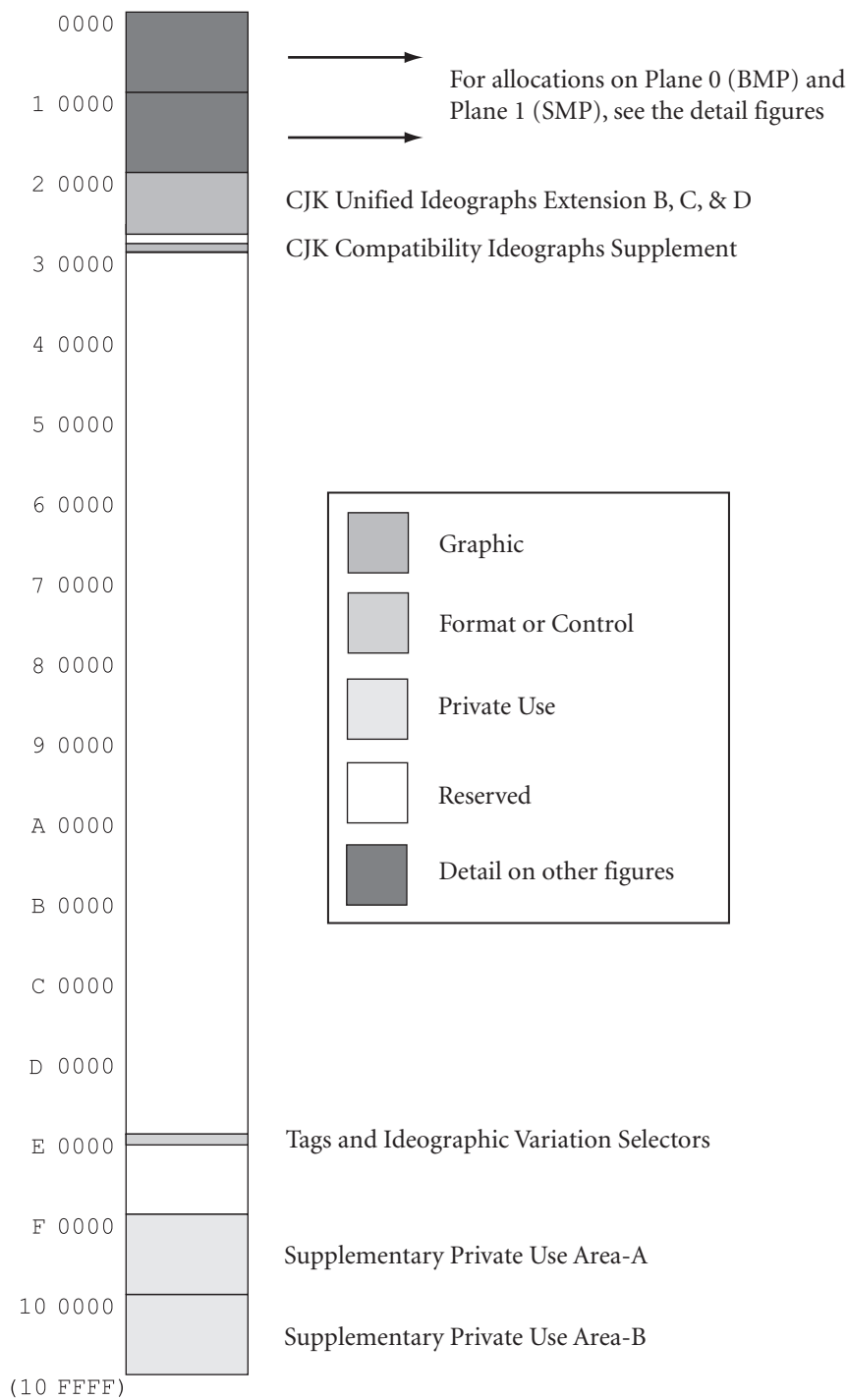
Code points in the Unicode Standard are assigned using the following guidelines:

- 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..nn3FE, nn400..nn7FE, nn800..nnBFE, 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 four sections, each of which is arranged according to the Han ideograph arrangement defined in *Section 12.1, Han*. This ordering is roughly based on a radical-stroke count order.

2.9 Details of Allocation

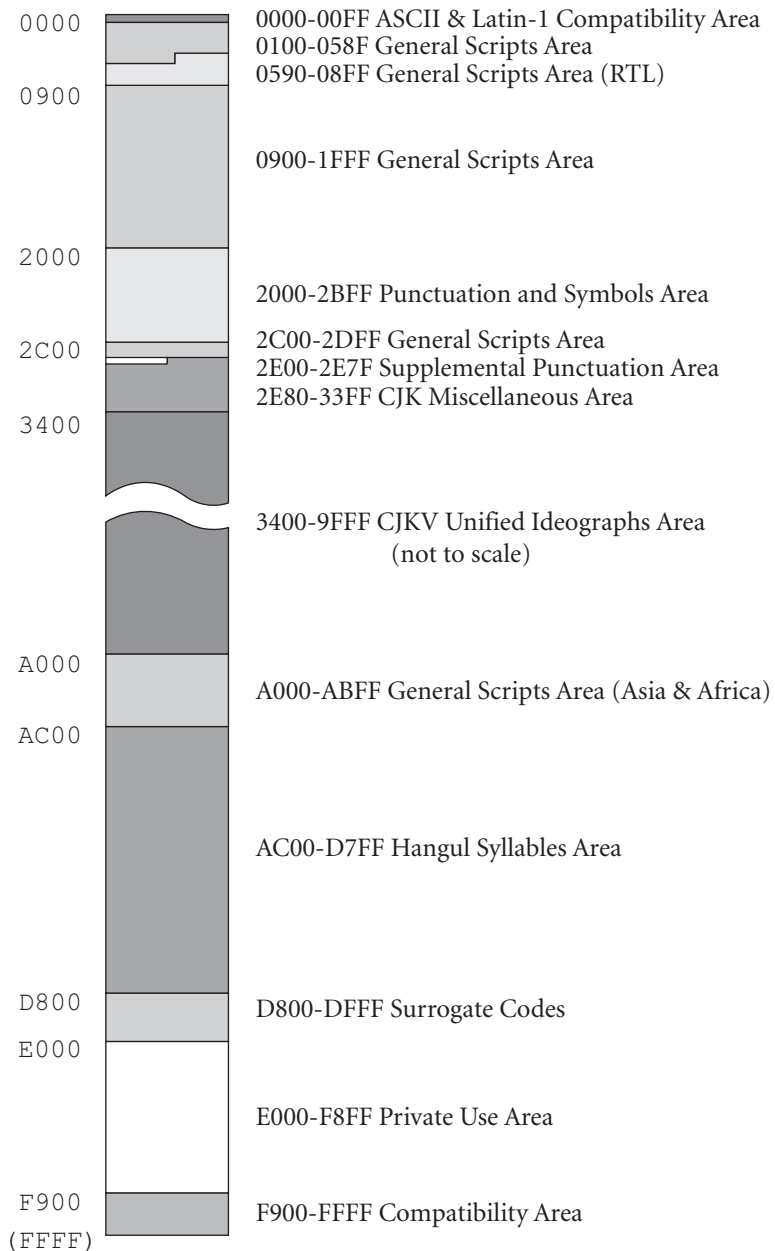
This section provides a more detailed summary of the way characters are allocated in the Unicode Standard. *Figure 2-13* gives an overall picture of the allocation areas of the Unicode Standard, with an emphasis on the identities of the planes. The following subsections discuss the allocation details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

Figure 2-13. Unicode Allocation



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

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 38 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 spill-over 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 16.6, Surrogates Area*, for more details.

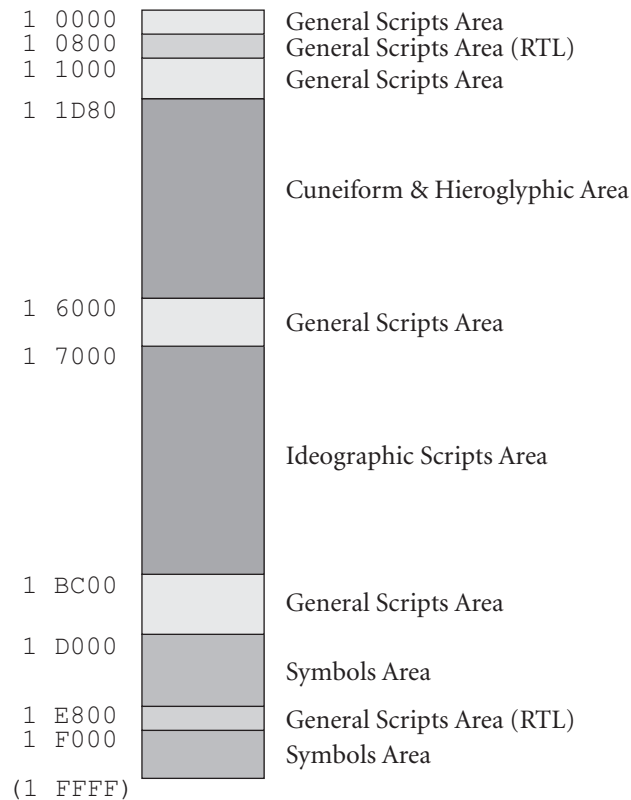
Private Use Area. The Private Use Area in the BMP contains 6,400 private-use characters.

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 a few important format control characters and other special characters. See *Section 16.8, Specials*, for more details.

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



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 currently has no assigned characters, but it 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 two large, ancient scripts: Sumero-Akkadian Cuneiform and Egyptian 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, Khitan, and Naxi. As of Unicode 6.1 no characters are yet encoded in this area.

Symbols Areas. The first of these SMP Symbols Areas contains sets of symbols for notational systems, such as musical symbols 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.

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.

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 language 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.

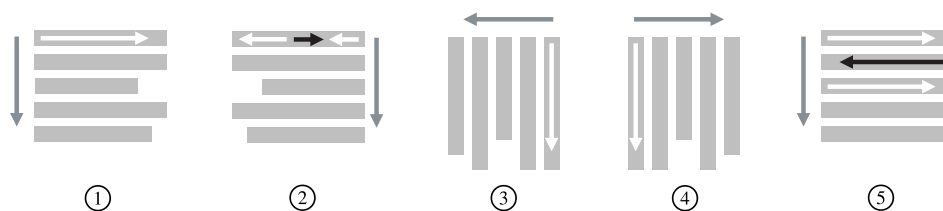
Plane 15 and Plane 16 are allocated, in their entirety, for private use. Those two planes contain a total of 131,068 characters, to supplement the 6,400 private-use characters located in the BMP.

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 and Noncharacters](#)).

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.

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. Characters intended to be positioned relative to an associated base character are depicted in the character code charts above, below, or through a 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 two types of combining characters: spacing and nonspacing. 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 “ı” may be slightly wider than a plain “i”. The spacing or nonspacing properties of a combining character are defined 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.*

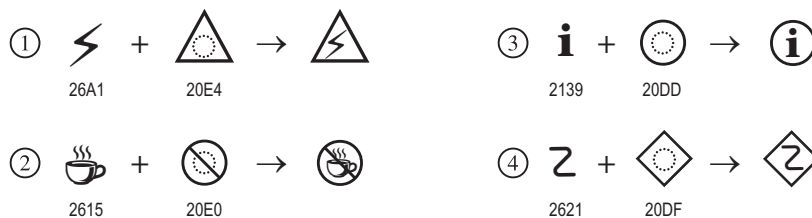
This does not create an obligation on implementations to support all possible combinations equally well. Thus, while application of an Arabic annotation mark to a Han character or a Devanagari consonant is permitted, it is unlikely to be supported well in rendering or to make much sense.

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 encode symbols. A few symbol characters are intended primarily for use with enclosing combining marks. For example, U+2139 INFORMATION SOURCE is a symbol intended for use with U+20DD COMBINING ENCLOSING CIRCLE or U+20E2 COMBINING ENCLOSING SCREEN. 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.

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*.

Sequence of Base Characters and Diacritics

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 “ö” COMBINING DIAERESIS, U+0075 “u” LATIN SMALL LETTER U> unambiguously represents “äü” and not “aü”, as shown in Figure 2-18.

Figure 2-18. Sequence of Base Characters and Diacritics

a + ö + u → äü (not aü)
0061 0308 0075

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 + ि → फि
092B 093F

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

! + △ → △ ≠ △
0021 20E4 26A0

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.

Multiple Combining Characters

In some instances, more than one diacritical mark 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. Stacking Sequences

Characters	Glyphs
a + ¨ + ˜ + ◌ + ◌ <small>0061 0308 0303 0323 032D</small>	→ ä̃̇̈
३ + ◌ + ◌ <small>0E02 0E36 0E49</small>	→ ३̂̃̄

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.

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
â	LATIN SMALL LETTER A WITH DOT ABOVE LATIN SMALL LETTER A + COMBINING DOT ABOVE
ạ̃	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
ậ	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 LATIN SMALL LETTER A + COMBINING CIRCUMFLEX + COMBINING ACUTE
â̂	LATIN SMALL LETTER A ACUTE + COMBINING CIRCUMFLEX LATIN SMALL LETTER A + COMBINING ACUTE + COMBINING CIRCUMFLEX



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

	GREEK SMALL LETTER ALPHA + COMBINING COMMA ABOVE (psili) + COMBINING ACUTE ACCENT (oxia)	This is correct
	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 of positioning of multiple points and marks in *Section 8.1, Hebrew*, or the discussion of nondefault placement of Arabic vowel marks accompanying *Figure 8-5 in Section 8.2, Arabic*.

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.

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

$$\begin{array}{ccccccc}
 \mathbf{f} & + & \tilde{\circ} & + & \mathbf{i} & + & \circ \\
 0066 & & 0303 & & 0069 & & 0323
 \end{array}
 \rightarrow
 \tilde{\mathbf{fi}}$$

For more information, see “Application of Combining Marks” in *Section 3.11, Normalization Forms*.

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.

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 Diacritics” in *Section 7.9, Combining Marks*.

“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”, tri-graphs 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*, and it consists of any combining character sequence that contains only *nonspacing* combining marks or any sequence of characters that constitutes a Hangul syllable (possibly followed by one or more nonspacing marks). 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 and Normalization

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.

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

Figure 2-23. Equivalent Sequences

$$\begin{array}{l}
 \textcircled{1} \quad \underset{0042}{\text{B}} + \underset{00C4}{\text{Ä}} \equiv \underset{0042}{\text{B}} + \underset{0041}{\text{A}} + \underset{0308}{{\text{̇}}{\text{̇}}} \\
 \textcircled{2} \quad \underset{01C7}{\text{LJ}} + \underset{0041}{\text{A}} \approx \underset{004C}{\text{L}} + \underset{004A}{\text{J}} + \underset{0041}{\text{A}} \\
 \textcircled{3} \quad \underset{0032}{\text{2}} + \underset{00BC}{\text{1/4}} \approx \underset{0032}{\text{2}} + \underset{0031}{\text{1}} + \underset{2044}{\text{/}} + \underset{0034}{\text{4}}
 \end{array}$$

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.”

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

$$\begin{array}{l}
 \textcircled{1} \quad \underset{0041}{\text{A}} + \overset{\text{non-interacting}}{\text{̇}} + \underset{0328}{\text{̇}} \equiv \underset{0041}{\text{A}} + \underset{0328}{\text{̇}} + \underset{0301}{\text{̇}} \\
 \text{ccc}=0 \quad \text{ccc}=230 \quad \text{ccc}=220 \qquad \text{ccc}=0 \quad \text{ccc}=220 \quad \text{ccc}=230 \\
 \textcircled{2} \quad \underset{0041}{\text{A}} + \overset{\text{interacting}}{\text{̇}} + \underset{0308}{{\text{̇}}{\text{̇}}} \neq \underset{0041}{\text{A}} + \underset{0308}{{\text{̇}}{\text{̇}}} + \underset{0301}{\text{̇}} \\
 \text{ccc}=0 \quad \text{ccc}=230 \quad \text{ccc}=230 \qquad \text{ccc}=0 \quad \text{ccc}=230 \quad \text{ccc}=230
 \end{array}$$

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.

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 *non-decomposable*.

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.

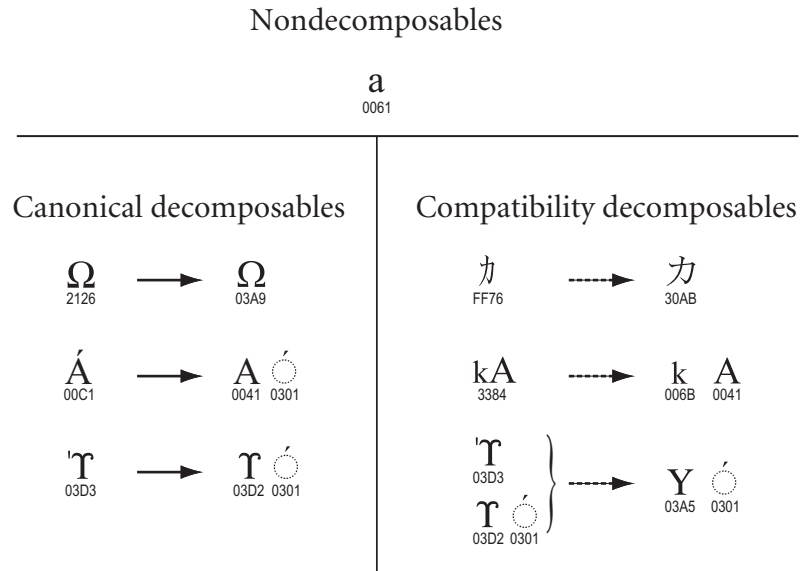
Examples. *Figure 2-25* illustrates these three types. Compatibility decompositions that are redundant because they are identical to the canonical decompositions are not shown. The figure illustrates two important points:

- 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 *Figure 2-25*. 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 *Figure 2-25* is for the Greek hooked upsi-

ion 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 *Chapter 3, Conformance*.

Figure 2-25. Types of Decomposables



Non-decomposition of Overlaid 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 *Figure 2-25*.

Based on that 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. Such characters include such 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.

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

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.

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.

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 and Noncharacters

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 16.1, Control Codes*; *Section 16.2, Layout Controls*; *Section 16.7, Noncharacters*; and *Section 16.8, Specials*.

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 should never be used for open interchange of Unicode text. For more information on noncharacters, see *Section 16.7, Noncharacters*.

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 con-

trol 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 FE₁₆ FF₁₆ (or its byte-reversed counterpart, FF₁₆ FE₁₆) is exceedingly rare at the outset of text files that use other character encodings. The corresponding UTF-8 BOM sequence, EF₁₆ BB₁₆ BF₁₆, 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. It is recommended that applications sending or receiving untyped data streams of coded characters use this signature. If other signaling methods are used, signatures should not be employed.

Conformance to the Unicode Standard does not require the use of the BOM as such a signature. See *Section 16.8, Specials*, for more information on the byte order mark and its use as an encoding signature.

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 16.2, Layout Controls*.

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 16.8, Specials*).

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 16.1, 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. Neverthe-

less, 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*.

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* ‘ゝ’ hiragana small i (which is the meaning of the bytes 2423₁₆ in JIS).
- U+00F4 ‘ô’ is equivalent to U+006F ‘o’ followed by U+0302 ◌̂, but *not equivalent* to U+0302 followed by U+006F.
- U+05D0 ‘ס’ followed by U+05D1 ‘ב’ looks like ‘סב’, *not* ‘בס’ 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.)

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.

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.

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.

A number of clauses and definitions have been updated from their wording in prior versions of the Unicode Standard. A detailed listing of these changes since Version 5.0, as well as a listing of any new definitions added, is available in *Section D.2, Clause and Definition Updates*.

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 update 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.

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 Web site. See the subsection “Policies” in *Section B.6, 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 Down-level Systems” in *Section 5.3, Unknown and Missing Characters*.

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.

Additional information on the current and past versions of the Unicode Standard can be found on the Unicode Web site. See the subsection “Versions” in *Section B.6, 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 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, because of the differences in the way versions are documented, update versions generally do not obsolete the documentation of the immediately prior version of the standard.

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 Web site. See *Section B.6, 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 *Section B.6, 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 *Section B.6, Other Unicode Online Resources*.

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 6.1.0, of the Unicode Standard, is

The Unicode Consortium. The Unicode Standard, Version 6.1.0, defined by: *The Unicode Standard, Version 6.1* (Mountain View, CA: The Unicode Consortium, 2012. ISBN 978-1-936213-02-3)

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 *Section B.6, Other Unicode Online Resources*.

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.

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 6.1.0 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.

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 6.1.0 of the Unicode Standard.

See Unicode Standard Annex #9, “Unicode Bidirectional Algorithm,” (<http://www.unicode.org/reports/tr9/tr9-25.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 Web site 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 Version 6.1 of the Unicode Standard.

In addition to this core specification, the Unicode Standard, Version 6.1.0, 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 6.1.0, 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.

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.

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 9.1, Devanagari*, or the rules about Arabic character shaping specified in *Section 8.2, 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 *Section 5.12, Strategies for Handling Nonspacing Marks*).

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 16.7, 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.”

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 110xxxx₂ *must* be followed by a code unit of the form 10xxxxx₂. A sequence such as 110xxxx₂ 0xxxxxx₂ 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 110xxxx₂ as an illegally terminated code unit sequence—for example, by signaling an error, filtering the code unit out, or representing the code unit with a marker such as U+FFFD REPLACEMENT CHARACTER.
 - 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.”

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.)

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.”

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.

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.

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.

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.

Unicode Standard Annexes

The following standard annexes are approved and considered part of Version 6.1 of the Unicode Standard. These annexes may contain either normative or informative material, or both. Any reference to Version 6.1 of the standard automatically includes these standard annexes.

- UAX #9: Unicode Bidirectional Algorithm, Version 6.1.0
- UAX #11: East Asian Width, Version 6.1.0
- UAX #14: Unicode Line Breaking Algorithm, Version 6.1.0
- UAX #15: Unicode Normalization Forms, Version 6.1.0
- UAX #24: Unicode Script Property, Version 6.1.0
- UAX #29: Unicode Text Segmentation, Version 6.1.0
- UAX #31: Unicode Identifier and Pattern Syntax, Version 6.1.0
- UAX #34: Unicode Named Character Sequences, Version 6.1.0
- UAX #38: Unicode Han Database (Unihan), Version 6.1.0
- UAX #41: Common References for Unicode Standard Annexes, Version 6.1.0
- UAX #42: Unicode Character Database in XML, Version 6.1.0
- UAX #44: Unicode Character Database, Version 6.1.0

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

Definitions

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

A small number of definitions have been updated from their wording in Version 5.0 of the Unicode Standard. A detailed listing of these changes, as well as a listing of any new definitions added since Version 5.0, is available in *Section D.2, Clause and Definition Updates*.

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 16.4, 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 17.1, 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 identity, 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, and named character sequences.
- 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 17.1, 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, and named character sequences 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 10FFFF₁₆.

- 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.

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 sequence is 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* alone designates 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 tag characters (see *Section 16.9, Deprecated Tag Characters*) and the alternate format characters (see *Section 16.3, 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 for internal use and that should never be interchanged. Noncharacters consist of the values $U+n\text{FFFE}$ and $U+n\text{FFFF}$ (where n is from 0 to 10_{16}) and the values $U+\text{FDD0}..U+\text{FDEF}$.

- For more information, see *Section 16.7, Noncharacters*.
- These code points are permanently reserved as 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	<i>Section 3.11</i>
Canonical Composition	<i>Section 3.11</i>
Normalization	<i>Section 3.11</i>
Hangul Syllable Composition	<i>Section 3.12</i>
Hangul Syllable Decomposition	<i>Section 3.12</i>
Hangul Syllable Name Generation	<i>Section 3.12</i>
Default Case Conversion	<i>Section 3.13</i>
Default Case Detection	<i>Section 3.13</i>
Default Caseless Matching	<i>Section 3.13</i>
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
Default Identifier Determination	UAX #31
Alternative Identifier Determination	UAX #31
Pattern Syntax Determination	UAX #31
Identifier Normalization	UAX #31
Identifier Case Folding	UAX #31
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.

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=C}`s 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.

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.

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 16.5, 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 defined implicitly, to avoid having to repeat long lists of unassigned code points.
- 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.

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.

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.”

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. Nevertheless, 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 <i>Section 4.4</i>
Bidi_Mirrored	UAX #9 and <i>Section 4.7</i>
Block	<i>Section 17.1</i>
Canonical_Combining_Class	<i>Section 3.11</i> and <i>Section 4.3</i>
Case-related properties	<i>Section 3.13</i> , <i>Section 4.2</i> , and UAX #44
Composition_Exclusion	<i>Section 3.11</i>
Decomposition_Mapping	<i>Section 3.7</i> and <i>Section 3.11</i>
Default_Ignorable_Code_Point	<i>Section 5.21</i>
Deprecated	<i>Section 3.1</i>
General_Category	<i>Section 4.5</i>
Hangul_Syllable_Type	<i>Section 3.12</i> and UAX #29
Joining_Type and Joining_Group	<i>Section 8.2</i>
Name	<i>Section 4.8</i>
Noncharacter_Code_Point	<i>Section 16.7</i>
Numeric_Value	<i>Section 4.6</i>
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
East_Asian_Width	Section 12.4 and UAX #11
Letter-related properties	Section 4.10
Line_Break	Section 16.1, Section 16.2, and UAX #14
Mathematical	Section 15.5
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 4-13 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.

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.

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 *Section B.6, Other Unicode Online Resources*.

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.

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 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”).

Private Use

D49 Private-use code point: Code points in the ranges U+E000..U+F8FF, U+F000..U+FFFFD, and U+100000..U+10FFFFD.

- 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 be 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 16.5, Private-Use Characters*.

3.6 Combination

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-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 (“i”), 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>.

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.)

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, U+FF9F, 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.

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.”

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 does not preclude spacing marks or even sequences of base characters from having effects on neighboring characters in rendering. Thus 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 9, South Asian Scripts-I*, for many examples.) Similarly, sequences of base characters may form ligatures in rendering. (See “Cursive Connection and Ligatures” in *Section 16.2, Layout Controls*.)

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 “äü”, not “aü”. See *Figure 2-18*.

P2 [Guideline] Inside-out application. Nonspacing marks with the same combining class 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.
- 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.
- 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*.
- 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.

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 8-5*.

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.

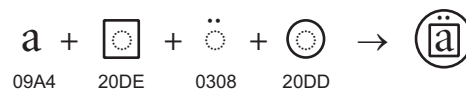
- For example, if one nonspacing mark occurs above a grapheme base and another nonspacing mark occurs below it, they will have distinct combining classes. The order in which they occur in the combining character sequence does not matter for the display or interpretation of the resulting grapheme cluster.

- 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 16.2, Layout Controls*.
- 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*.

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 7-8* 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 7-9* 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 7-10* 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 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 grave is applied to the entire Korean syllable, not just to the last jamo:

U+1100 ㄱ *choseong kiyeok* + U+1161 ㅈ *jungseong a* + U+0300 ◌ *grave* →
 ㄱㅈ

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 ㅈ *jungseong a* + U+20DD ◉ *enclosing circle* → (ㄱㅈ)

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 16.2, *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 17.1, *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 17.1, *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 decom-

posing each of the characters in the sequence until no characters can be further decomposed.

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 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 given in *Section 17.1, Character Names List*.
- Some compatibility decompositions remove formatting information.

D66 *Compatibility decomposable character*: A character whose compatibility decomposition 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.6, 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.

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 17.1, 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 D800₁₆ to DBFF₁₆, 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 DC00₁₆ to DFFF₁₆, 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 16.6, 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 *Section 5.2, Programming Languages and Data Types*.

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 A *Unicode encoding form* assigns each Unicode scalar value to a unique code unit sequence.

- 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 *Table 3-7*.
- 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 subse-

quence <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

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 0000D800₁₆..0000DFFF₁₆ are ill-formed.
- Any UTF-32 code unit greater than 0010FFFF₁₆ is ill-formed.

For a discussion of the relationship between UTF-32 and UCS-4 encoding form defined in ISO/IEC 10646, see *Section C.2, Encoding Forms in ISO/IEC 10646*.

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 D800₁₆..DFFF₁₆ 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 10000₁₆, to account for the starting

offset to the scalar value. ISO/IEC 10646 specifies an equivalent UTF-16 encoding form. For details, see *Section C.3, UTF-8 and UTF-16*.

Table 3-5. UTF-16 Bit Distribution

Scalar Value	UTF-16
xxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxx
000uuuuuxxxxxxxxxxxxxxxx	110110wwwxxxxxxxx 11011xxxxxxxx

Note: www = uuuu - 1

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 D800..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 *Section 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		
zzzyyyyy 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	<i>A0..BF</i>	80..BF	
U+1000..U+CFFF	E1..EC	80..BF	80..BF	
U+D000..U+D7FF	ED	80.. <i>9F</i>	80..BF	
U+E000..U+FFFF	EE..EF	80..BF	80..BF	
U+10000..U+3FFFF	F0	<i>90..BF</i>	80..BF	80..BF
U+40000..U+FFFFF	F1..F3	80..BF	80..BF	80..BF
U+100000..U+10FFFF	F4	80.. <i>8F</i>	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.

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.

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 uninterpreta-

ble, 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.

Best Practices for Using U+FFFD. When using U+FFFD to replace ill-formed subsequences encountered during conversion, there are various logically possible approaches to associate U+FFFD with all or part of an ill-formed subsequence. To promote interoperability in the implementation of conversion processes, the Unicode Standard recommends a particular best practice. The following definitions simplify the discussion of this best practice:

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.
- The term *maximal subpart of an ill-formed subsequence* can be abbreviated to *maximal subpart* when it is clear in context that the subsequence in question is ill-formed.
 - This definition can be trivially applied to the UTF-32 or UTF-16 encoding forms, but is primarily of interest when converting UTF-8 strings.
 - For example, in the ill-formed UTF-8 sequence <41 C0 AF 41 F4 80 80 41>, there are two ill-formed subsequences: <C0 AF> and <F4 80 80>, each separated by <41>, which is well-formed. Applying the definition of maximal subparts for these ill-formed subsequences, in the first case <C0> is a maximal subpart, because that byte value can never be the first byte of a well-formed UTF-8 sequence. In the second subsequence, <F4 80 80> is a maximal subpart, because up to that point all three bytes match the specification for UTF-8. It is only when followed by <41> that the sequence of <F4 80 80> can be determined to be ill-formed, because the specification requires a following byte in the range 80..BF, instead.
 - Another example illustrates the application of the concept of maximal subpart for UTF-8 continuation bytes outside the allowable ranges defined in *Table 3-7*. The UTF-8 sequence <41 E0 9F 80 41> is ill-formed, because <9F> is not an

allowed second byte of a UTF-8 sequence commencing with <E0>. In this case, there is an unconvertible offset at <E0> and the maximal subpart at that offset is also <E0>. The subsequence <E0 9F> cannot be a maximal subpart, because it is not an initial subsequence of any well-formed UTF-8 code unit sequence.

Using the definition for maximal subpart, the best practice can be stated simply as:

Whenever an unconvertible offset is reached during conversion of a code unit sequence:

1. *The maximal subpart at that offset should be replaced by a single U+FFFD.*
2. *The conversion should proceed at the offset immediately after the maximal subpart.*

This sounds complicated, but it reflects the way optimized conversion processes are typically constructed, particularly for UTF-8. A sequence of code units will be processed up to the point where the sequence either can be unambiguously interpreted as a particular Unicode code point or where the converter recognizes that the code units collected so far constitute an ill-formed subsequence. At that point, the converter can emit a single U+FFFD for the collected (but ill-formed) code unit(s) and move on, without having to further accumulate state. The maximal subpart could be the start of a well-formed sequence, except that the sequence lacks the proper continuation. Alternatively, the converter may have found a continuation code unit or some other code unit which cannot be the start of a well-formed sequence.

To illustrate this policy, consider the ill-formed UTF-8 sequence <61 F1 80 80 E1 80 C2 62 80 63 80 BF 64>. Possible alternative approaches for a UTF-8 converter using U+FFFD are illustrated in *Table 3-8*.

Table 3-8. Use of U+FFFD in UTF-8 Conversion

	61	F1	80	80	E1	80	C2	62	80	63	80	BF	64
1	0061	FFFD						0062	FFFD	0063	FFFD		0064
2	0061	FFFD			FFFD		FFFD	0062	FFFD	0063	FFFD	FFFD	0064
3	0061	FFFD	FFFD	FFFD	FFFD	FFFD	FFFD	0062	FFFD	0063	FFFD	FFFD	0064

The recommended conversion policy would have the outcome shown in Row 2 of *Table 3-8*, rather than Row 1 or Row 3. For example, a UTF-8 converter would detect that <F1 80 80> constituted a maximal subpart of the ill-formed subsequence as soon as it encountered the subsequent code unit <E1>, so at that point, it would emit a single U+FFFD and then continue attempting to convert from the <E1> code unit—and so forth to the end of the code unit sequence to convert. The UTF-8 converter would detect that the code unit <80> in the sequence <62 80 63> is not well-formed, and would replace it by U+FFFD. Neither of the code units <80> or <BF> in the sequence <63 80 BF 64> is the start of a potentially well-formed sequence; therefore *each* of them is separately replaced by U+FFFD. For a discussion of the generalization of this approach for conversion of other character sets to Unicode, see *Section 5.22, Best Practice for U+FFFD Substitution*.

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 neither required nor recommended 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 3-9 gives examples that summarize the three Unicode encoding schemes for the UTF-16 encoding form.

Table 3-9. 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
	UTF-16	FE FF 4E 8C FF FE 8C 4E 4E 8C
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 3-10 gives examples that summarize the three Unicode encoding schemes for the UTF-32 encoding form.

Table 3-10. 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.]

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 *Section B.6, 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.”

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 alternate 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 `\p{Canonical_Combining_Class = 0}` (or `\p{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 above are given the value `\p{ccc=230}`, while all generic nonspacing marks below are given the value `\p{ccc=220}`. Smaller numbers of 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.

When assigned this way, canonical ordering assures that, in general, alternate sequences of combining characters that typographically interact will not be canonically equivalent,

whereas alternate sequences of combining characters that do *not* typographically interact *will* be canonically equivalent.

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.

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.

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 3-11* summarizes the relationship between types of combining marks and their status as Starters.

Table 3-11. 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.

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 3-12 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 3-12. 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	$ccc(A) > ccc(B)$

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 Decomposition 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 $ccc(B) \geq ccc(C)$.

- 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.

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 12.6, 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.”

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 HANGUL JUNGSEONG FILLER. 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 $\langle L, V \rangle$.

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 $\langle LV, T \rangle$.

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.

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 3-13*. Finally, each algorithm is then further exemplified with an implementation as a Java method at the end of this section.

Table 3-13. Hangul Characters Used in Examples

Code Point	Glyph	Character Name	Jamo Short Name
U+D4DB	ㅍㅅ	HANGUL SYLLABLE PWILH	
U+1111	ㅍ	HANGUL CHOSEONG PHIEUPH	P
U+1171	ㅍ	HANGUL JUNGSEONG WI	WI
U+11B6	ㅍ	HANGUL JONGSEONG RIEUL-HIEUH	LH

Common Constants. Define the following consonants:

```

SBase = AC0016
LBase = 110016
VBase = 116116
TBase = 11A716
SCount = 11172
LCount = 19
VCount = 21
TCount = 28
NCount = 588 (VCount * TCount)
SCount = 1172 (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 NCount) * NCount
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 `<LVPart, 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
LVPart = 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 $d = \langle LVPart, VPart \rangle$. Otherwise, there is a trailing consonant, so map s to its full decomposition $d = \langle LVPart, VPart, TPart \rangle$.

Example. For the precomposed Hangul syllable `U+D4DB`, compute the indices and components:

```

SIndex = 10459
LIndex = 17
VIndex = 16
TIndex = 15
LVPart = 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.

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 $\langle \text{LPart}, \text{VPart} \rangle$, 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:

$$\begin{aligned} \text{LIndex} &= \text{LPart} - \text{LBase} \\ \text{VIndex} &= \text{VPart} - \text{VBase} \\ \text{LVIndex} &= \text{LIndex} * \text{NCount} + \text{VIndex} * \text{TCount} \\ s &= \text{SBase} + \text{LVIndex} \end{aligned}$$

Given a Hangul jamo sequence $\langle \text{LPart}, \text{VPart}, \text{TPart} \rangle$, 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:

$$\begin{aligned} \text{LIndex} &= \text{LPart} - \text{LBase} \\ \text{VIndex} &= \text{VPart} - \text{VBase} \\ \text{TIndex} &= \text{TPart} - \text{TBase} \\ \text{LVIndex} &= \text{LIndex} * \text{NCount} + \text{VIndex} * \text{TCount} \\ s &= \text{SBase} + \text{LVIndex} + \text{TIndex} \end{aligned}$$

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 $\langle \text{LVPart}, \text{TPart} \rangle$, 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:

$$\begin{aligned} \text{TIndex} &= \text{TPart} - \text{TBase} \\ s &= \text{LVPart} + \text{TIndex} \end{aligned}$$

Example. For the canonically decomposed Hangul jamo sequence $\langle \text{U+1111}, \text{U+1171}, \text{U+11B6} \rangle$, compute the indices and syllable mapping:

$$\begin{aligned} \text{LIndex} &= 17 \\ \text{VIndex} &= 16 \\ \text{TIndex} &= 15 \\ \text{LVIndex} &= 17 * 588 + 16 * 28 = 9996 + 448 = 10444 \\ s &= \text{AC00}_{16} + 10444 + 15 = \text{D4DB}_{16} \end{aligned}$$

Then map the Hangul jamo sequence to this precomposed Hangul syllable as its Primary Composite:

$$\langle \text{U+1111}, \text{U+1171}, \text{U+11B6} \rangle \rightarrow \text{U+D4DB}$$

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:

$$s \rightarrow d = \langle \text{LPart}, \text{VPart}, (\text{TPart}) \rangle$$

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 " + JSNL + JSNV + JSNT
```

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:

```
JSNL = Jamo_Short_Name(U+1111) = "P"
JSNV = Jamo_Short_Name(U+1171) = "WI"
JSNT = Jamo_Short_Name(U+11B6) = "LH"
```

Concatenate the pieces:

```
Name = "HANGUL SYLLABLE " + "P" + "WI" + "LH"
      = "HANGUL SYLLABLE PWILH"
```

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);
    }
    StringBuffer result = new StringBuffer();
    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 SpecialCasing.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), 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*.

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` or the value `MidNumLet` 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 3-14*. 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 *Case-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 3-14*.

Table 3-14. 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	<code>\p{cased} (\p{case-ignorable})*</code>
		After C	<code>! ((\p{case-ignorable})* \p{cased})</code>
After_Soft_Dotted	There is a <code>Soft_Dotted</code> character before C, with no intervening character of combining class 0 or 230 (Above).	Before C	<code>[\p{Soft_Dotted}] ([^\p{ccc=230} \p{ccc=0}])*</code>
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	<code>[^\p{ccc=230} \p{ccc=0}]* [\p{ccc=230}]</code>
Before_Dot	C is followed by <code>COMBINING DOT ABOVE</code> (U+0307). Any sequence of characters with a combining class that is neither 0 nor 230 may intervene between the current character and the <i>combining dot above</i> .	After C	<code>([^\p{ccc=230} \p{ccc=0}])* [\u0307]</code>
After_I	There is an uppercase I before C, and there is no intervening combining character class 230 (Above) or 0.	Before C	<code>[I] ([^\p{ccc=230} \p{ccc=0}])*</code>

In *Table 3-14*, 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 3-14* 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 `YPÖGEGRAMMENI`. Thus, the `Before` condition is not satisfied if C is preceded by only U+0345, but would be satisfied by the sequence `<capital-alpha, ypögegrammeni>`. Similarly, the `After` condition is satisfied if C is only followed by `ypögegrammeni`, but would not be satisfied by the sequence `<ypögegrammeni, capital-alpha>`.

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 3-14*.

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.

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.

Unicode Default Case Folding is built on the `toLowercase(X)` transform, with some adaptations specifically for caseless matching. Context-dependent mappings based on the casing context are not used.

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 case-folded.

Default Case Folding is based on the full case conversion operation, `Lowercase_Mapping`, which includes conversions to lowercase forms that may change string length, but is adapted specifically for caseless matching. In particular, 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” or “F” in the data file `CaseFolding.txt` in the Unicode Character Database.

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.

For more information on the use of NFKC_Casefold and caseless matching for identifiers, see Unicode Standard Annex #31, “Unicode Identifier and Pattern Syntax.”

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 3-15* 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 3-15. 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)`).

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:

$$\text{NFD}(\text{toCasefold}(\text{NFD}(X))) = \text{NFD}(\text{toCasefold}(\text{NFD}(Y)))$$

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 η 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:

$$\text{NFKD}(\text{toCasefold}(\text{NFKD}(\text{toCasefold}(\text{NFD}(X)))))) = \text{NFKD}(\text{toCasefold}(\text{NFKD}(\text{toCasefold}(\text{NFD}(Y))))))$$

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:

$$\text{toNFKC_Casefold}(\text{NFD}(X)) = \text{toNFKC_Casefold}(\text{NFD}(Y))$$

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 Web site. *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 prop-

erties. 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 *Section B.6, Other Unicode Online Resources*.)

Many properties apply to both ideographs and other characters. These are not specified in the Unihan Database.

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 Web site. See the subsection “Policies” in *Section B.6, 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 Web site. See the subsections “Online Unicode Character Database” and “Online Unihan Database” in *Section B.6, 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 LATIN CAPITAL LETTER DZ, 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 LATIN CAPITAL LETTER D WITH SMALL LETTER Z. The three case forms are UPPERCASE, Titlecase, and lowercase.

For those scripts that have case (Latin, Greek, Coptic, Cyrillic, Glagolitic, Armenian, Deseret, and archaic Georgian), 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.*)

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, letter-like 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	Li	True	False	True	False
0048	H	Lu	False	True	False	True
24D7	⓪	So	True	False	True	False
24BD	Ⓜ	So	False	True	False	True
02B0	h	Lm	True	False	True	True
1D34	Ḥ	Lm	True	False	True	True
02BD	ˆ	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.

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
<code>UnicodeData.txt</code>	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.
<code>SpecialCasing.txt</code>	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.
<code>CaseFolding.txt</code>	Contains data for performing locale-independent case folding, as described in “Caseless Matching,” in <i>Section 5.18, Case Mappings</i> .
<code>PropList.txt</code>	Contains the definition of the property <code>Soft_Dotted</code> , which is used in the context specification for casing. See D138 in <i>Section 3.13, Default Case Algorithms</i> .

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 Web site. See “Charts” in *Section B.6, 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.

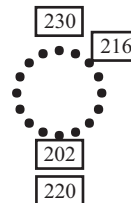
The canonical order of character sequences does *not* imply any kind of linguistic correctness or linguistic preference for ordering of combining marks in sequences. For more information on rendering combining marks, see *Section 5.13, Rendering Nonspacing Marks*.

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 impor-

tance 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



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, 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.

Such matters are not treated as normative character properties in the Unicode Standard, because they are more properly considered properties of the glyphs and fonts used for rendering. However, to assist implementers, this section subcategorizes some class zero combining marks for Brahmi-derived scripts, pointing out significant types that need to be handled consistently.

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 left 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.

Table 4-4 lists reordrant class zero combining marks in the Unicode Standard.

Table 4-4. Class Zero Combining Marks—Reordrant

Script	Code Points
Devanagari	093F, 094E
Bengali	09BF, 09C7, 09C8
Gurmukhi	0A3F
Gujarati	0ABF
Oriya	0B47
Tamil	0BC6, 0BC7, 0BC8
Malayalam	0D46, 0D47, 0D48
Sinhala	0DD9, 0DDA, 0ddb
Myanmar	1031, 1084
Khmer	17C1, 17C2, 17C3
New Tai Lue	19B5, 19B6, 19B7, 19BA
Buginese	1A19, 1A1B
Tai Tham	1A55, 1A6E, 1A6F, 1A70, 1A71, 1A72
Balinese	1B3E, 1B3F
Sundanese	1BA6
Lepcha	1C27, 1C28, 1C29, 1C34, 1C35
Javanese	A9BA, A9BB
Cham	AA2F, AA30, AA34
Meetei Mayek	AAEB, AAEE
Kaithi	110B1
Chakma	1112C
Sharada	11184
Takri	116AE

In addition, there are historically related vowel characters in the Thai, Lao, and Tai Viet scripts that, for legacy reasons, are not treated as combining marks. Instead, for Thai, Lao, and Tai Viet these 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 logically for searching and sorting. Because of that processing requirement, these characters are given a formal character property assignment, the `Logical_Order_Exception` property, as listed in Table 4-5. See `PropList.txt` in the Unicode Character Database.

Table 4-5. Thai, Lao, and Tai Viet Logical Order Exceptions

Script	Code Points
Thai	0E40..0E44
Lao	0EC0..0EC4
Tai Viet	AAB5, AAB6, AAB9, AABB, AABC

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.

Table 4-6 lists split class zero combining marks in the Unicode Standard, subgrouped by positional patterns.

Table 4-6. Class Zero Combining Marks—Split

Glyph Positions	Script	Code Points
Left and right	Bengali	09CB, 09CC
	Oriya	0B4B
	Tamil	0BCA, 0BCB, 0BCC
	Malayalam	0D4A, 0D4B, 0D4C
	Sinhala	0DDC, 0DDE
	Khmer	17C0, 17C4, 17C5
	Balinese	1B40, 1B41
Left and top	Oriya	0B48
	Sinhala	0DDA
	Khmer	17BE
Left, top, and right	Oriya	0B4C
	Sinhala	0DDD
	Khmer	17BF
Top and right	Oriya	0B57
	Kannada	0CC0, 0CC7, 0CC8, 0CCA, 0CCB
	Limbu	1925, 1926
	Balinese	1B43
Top and bottom	Telugu	0C48
	Tibetan	0F73, 0F76, 0F77, 0F78, 0F79, 0F81
	Balinese	1B3C
Top, bottom, and right	Balinese	1B3D
Bottom and right	Balinese	1B3B

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.

Table 4-7 lists subjoined class zero combining marks in the Unicode Standard.

These Limbu consonants, while logically considered subjoined combining marks, are rendered mostly at the lower right of a base letter, rather than directly beneath them.

Table 4-7. Class Zero Combining Marks—Subjoined

Script	Code Points
Tibetan	0F90..0F97, 0F99..0FBC
Limbu	1929, 192A, 192B

Strikethrough Class Zero Combining Marks. The Kharoshthi script is unique in having some class zero combining marks for vowels that are struck through a consonant, rather than being placed in a position around the consonant. These are also called out in *Table 4-8* specifically as a warning that they may involve particular problems for implementations.

Table 4-8. Class Zero Combining Marks—Strikethrough

Script	Code Points
Kharoshthi	10A01, 10A06

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 4-9 enumerates the General_Category values, giving a short description of each value. See *Table 2-3* for the relationship between General_Category values and basic types of code points.

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

Table 4-9. General Category

Lu = Letter, uppercase Ll = Letter, lowercase Lt = Letter, titlecase Lm = Letter, modifier Lo = Letter, other
Mn = Mark, nonspacing Mc = Mark, spacing combining Me = Mark, enclosing
Nd = Number, decimal digit Nl = Number, letter No = Number, other
Pc = Punctuation, connector Pd = Punctuation, dash Ps = Punctuation, open Pe = Punctuation, close Pi = Punctuation, initial quote (may behave like Ps or Pe depending on usage) Pf = Punctuation, final quote (may behave like Ps or Pe depending on usage) Po = Punctuation, other
Sm = Symbol, math Sc = Symbol, currency Sk = Symbol, modifier So = Symbol, other
Zs = Separator, space Zl = Separator, line Zp = Separator, paragraph
Cc = Other, control Cf = Other, format Cs = Other, surrogate Co = Other, private use Cn = Other, not assigned (including noncharacters)

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 SIGN 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.
- 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, circled 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 4-10*), 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 15.4, Superscript and Subscript Symbols*.) 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 15-3* for a list of script-specific digits. For naming conventions relevant to the Arabic digits, see the introduction to *Section 8.2, Arabic*.

Ideographic Numeric Values

CJK ideographs also may have numeric values. The primary numeric ideographs are shown in *Table 4-10*. When used to represent numbers in decimal notation, zero is represented by U+3007. Otherwise, zero is represented by U+96F6.

Table 4-10. Primary Numeric Ideographs

Code Point	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+5104	100,000,000 (10,000 × 10,000)
U+4EBF	100,000,000 (10,000 × 10,000)
U+5146	1,000,000,000,000 (10,000 × 10,000 × 10,000)

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 4-11* 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 4-11. 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 “)”.

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 expression 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.

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. Note that mirroring is not limited to paired characters, but that any character with the mirrored property will need two mirrored glyphs—for example, U+222B INTEGRAL. This requirement is necessary to render the character properly in a bidirectional context. It is the default behavior in Unicode text. (For more information, see the “Semantics of Paired Punctuation” subsection in *Section 6.2, General Punctuation*.)

This property is not to be confused with the related *Bidi Mirroring Glyph* property, an informative property, that can assist in rendering mirrored characters in a right-to-left context. For more information, see *BidiMirroring.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 *Section 15.9, Miscellaneous Symbols*.

Character names are listed in the code charts.

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+0056), 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 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 5.7, “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. Sometimes errors in a character name are discovered after publication. Because character names are immutable, such errors are not corrected by changing the names. However, in some limited instances (as for obvious typos in a character name), the Unicode Standard publishes an additional, corrected name as a normative *character name alias*. (See Definition D5 in Section 3.3, *Semantics*.) Character name aliases are immutable once published and are also guaranteed to be unique in the namespace for character names. A character may, in principle, have more than one normative character name alias.

Character name aliases which serve to correct errors in character names are listed in the code charts, using a special typographical convention explained in Section 17.1, *Character Names List*. They are also separately listed in the file `NameAliases.txt` in the Unicode Character Database.

In addition to such corrections, the file `NameAliases.txt` contains aliases that give definitive labels to control codes, which have no actual Unicode character name. Additional aliases match existing and widely used alternative names and abbreviations for control codes and for Unicode format characters. 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 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.

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.

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 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:

- For Hangul syllables, the Name property value is derived by rule, as specified in *Section 3.12, Conjoining Jamo Behavior*, under “Hangul Syllable Name Generation,” by combining a fixed prefix 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”.
- For ideographs, the Name property value is derived by concatenating the string “CJK UNIFIED IDEOGRAPH-” or “CJK COMPATIBILITY IDEOGRAPH-” to the code point, expressed in hexadecimal, with the usual 4- to 6-digit convention. For example, the name of U+4E00 is CJK UNIFIED IDEOGRAPH-4E00. Field 1 of the `UnicodeData.txt` data file uses a special convention to indicate the ranges of ideographs for which the Name property is derived by rule.
- For all other Graphic characters and for all Format characters, the Name property value is as listed in Field 1 of `UnicodeData.txt`. For example, U+0A15 GURMUKHI LETTER KA OR U+200D ZERO WIDTH JOINER.
- 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 generic term “character name” refers to the Name property value for an encoded Unicode character. An expression such as, “The reserved code point U+30000 has no name,” is shorthand for the more precise statement that the reserved code point U+30000 (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, and named character sequences constitute a single, unique namespace, the Name property value uniqueness requirement applies to all three kinds of names.

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*.)

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. The label construction for the five affected code point types is illustrated in Table 4-12.

Table 4-12. Construction of Code Point Labels

Type	Label
Control	control-NNNN
Reserved	reserved-NNNN
Noncharacter	noncharacter-NNNN
Private-Use	private-use-NNNN
Surrogate	surrogate-NNNN

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*.

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 specifi-

cation. 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, and characters from other blocks—for example, U+3007 IDEOGRAPHIC NUMBER ZERO and U+3006 IDEOGRAPHIC CLOSING MARK. For more information about Han ideographs, see *Section 12.1, Han*. For more about ideographs and logosyllabaries in general, see *Section 6.1, Writing Systems*.

4.11 Properties Related to 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 16.2, 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 4-13* 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.

Table 4-13. Unusual Properties

Function	Description	Code Point and Name
Fraction formatting	<i>Section 6.2</i>	2044 FRACTION SLASH
Special behavior with non-spacing marks	<i>Section 2.11, Section 6.2, and Section 16.2</i>	0020 SPACE 00A0 NO-BREAK SPACE
Double nonspacing marks	<i>Section 7.9</i>	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
Combining half marks	<i>Section 7.9</i>	FE20 COMBINING LIGATURE LEFT HALF FE21 COMBINING LIGATURE RIGHT HALF FE22 COMBINING DOUBLE TILDE LEFT HALF FE23 COMBINING DOUBLE TILDE RIGHT HALF FE24 COMBINING MACRON LEFT HALF FE25 COMBINING MACRON RIGHT HALF
Cursive joining and ligation control	<i>Section 16.2</i>	200C ZERO WIDTH NON-JOINER 200D ZERO WIDTH JOINER
Collation weighting and sequence interpretation	<i>Section 16.2</i>	034F COMBINING GRAPHEME JOINER
Bidirectional ordering	<i>Section 16.2</i>	200E LEFT-TO-RIGHT MARK 200F RIGHT-TO-LEFT MARK 202A LEFT-TO-RIGHT EMBEDDING 202B RIGHT-TO-LEFT EMBEDDING 202C POP DIRECTIONAL FORMATTING 202D LEFT-TO-RIGHT OVERRIDE 202E RIGHT-TO-LEFT OVERRIDE
Mathematical expression formatting	<i>Section 15.6</i>	2061 FUNCTION APPLICATION 2062 INVISIBLE TIMES 2063 INVISIBLE SEPARATOR 2064 INVISIBLE PLUS
Deprecated alternate formatting	<i>Section 16.3</i>	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
Prefixed format control	<i>Section 8.2, Section 8.3, and Section 10.7</i>	0600 ARABIC NUMBER SIGN 0601 ARABIC SIGN SANAH 0602 ARABIC FOOTNOTE MARKER 0603 ARABIC SIGN SAFHA 0604 ARABIC SIGN SAMVAT 06DD ARABIC END OF AYAH 070F SYRIAC ABBREVIATION MARK 110BD KATHI NUMBER SIGN

Table 4-13. Unusual Properties (Continued)

Function	Description	Code Point and Name
Brahmi-derived script dead-character formation	<i>Chapter 9, Chapter 10, and Chapter 11</i>	094D DEVANAGARI SIGN VIRAMA 09CD BENGALI SIGN VIRAMA 0A4D GURMUKHI SIGN VIRAMA 0ACD GUJARATI SIGN VIRAMA 0B4D ORIYA SIGN VIRAMA 0BCD TAMIL SIGN VIRAMA 0C4D TELUGU SIGN VIRAMA 0CCD KANNADA SIGN VIRAMA 0D4D MALAYALAM SIGN VIRAMA 0DCA SINHALA SIGN AL-LAKUNA 0E3A THAI CHARACTER PHINTHU 1039 MYANMAR SIGN VIRAMA 1714 TAGALOG SIGN VIRAMA 1734 HANUNOO SIGN PAMUDPOD 17D2 KHMER SIGN COENG 1A60 TAI THAM SIGN SAKOT 1B44 BALINESE ADEG ADEG 1BAA SUNDANESE SIGN PAMAAEH A806 SYLOTI NAGRI SIGN HASANTA A8C4 SAURASHTRA SIGN VIRAMA A953 REJANG VIRAMA A9C0 JAVANESE PANGKON AAF6 MEETEI MAYEK VIRAMA ABED MEETEI MAYEK APUN IYEK 10A3F KHAROSHTHI VIRAMA 110B9 KAITHI SIGN VIRAMA 11133 CHAKMA VIRAMA 111C0 SHARADA SIGN VIRAMA 116B6 TAKRI SIGN VIRAMA
Historical viramas with other functions	<i>Section 10.2 and Section 10.5</i>	0F84 TIBETAN MARK HALANTA 103A MYANMAR SIGN ASAT ABED MEETEI MAYEK APUN IYEK 193B LIMBU SIGN SA-I 11134 CHAKMA MAAYYAA
Mongolian variation selectors	<i>Section 13.2</i>	180B MONGOLIAN FREE VARIATION SELECTOR ONE 180C MONGOLIAN FREE VARIATION SELECTOR TWO 180D MONGOLIAN FREE VARIATION SELECTOR THREE 180E MONGOLIAN VOWEL SEPARATOR
Generic variation selectors	<i>Section 16.4</i>	FE00..FE0F VARIATION SELECTOR-1..VARIATION SELECTOR-16 E0100..E01EF VARIATION SELECTOR-17..VARIATION SELECTOR-256
Tag characters	<i>Section 16.9</i>	E0001 LANGUAGE TAG E0020..E007F LANGUAGE TAG SPACE..CANCEL TAG
Ideographic variation indication	<i>Section 6.2</i>	303E IDEOGRAPHIC VARIATION INDICATOR
Ideographic description	<i>Section 12.2</i>	2FF0..2FFB IDEOGRAPHIC DESCRIPTION CHARACTER LEFT TO RIGHT..IDEOGRAPHIC DESCRIPTION CHARACTER OVERLAID
Interlinear annotation	<i>Section 16.8</i>	FFF9 INTERLINEAR ANNOTATION ANCHOR FFFA INTERLINEAR ANNOTATION SEPARATOR FFFB INTERLINEAR ANNOTATION TERMINATOR
Object replacement	<i>Section 16.8</i>	FFFC OBJECT REPLACEMENT CHARACTER
Code conversion fallback	<i>Section 16.8</i>	FFFD REPLACEMENT CHARACTER

Table 4-13. Unusual Properties (Continued)

Function	Description	Code Point and Name
Musical format control	<i>Section 15.12</i>	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
Line break controls	<i>Section 16.2</i>	00AD SOFT HYPHEN 200B ZERO WIDTH SPACE 2060 WORD JOINER
Byte order signature	<i>Section 16.8</i>	FEFF ZERO WIDTH NO-BREAK SPACE

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.

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 $O(n)$.

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 disk space 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.

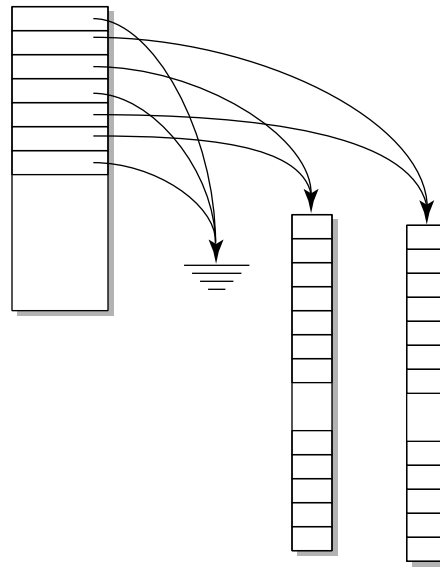
Ranges. It may be tempting to “optimize” these tables for space by providing elaborate provisions for nested ranges or similar devices. This practice leads to unnecessary performance costs on modern, highly pipelined processor architectures because of branch penalties. A faster solution is to use an *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.

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.

Figure 5-1. Two-Stage Tables



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.

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 `\Uhhhhhhh` 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, using appropriate glyphs such as  for reserved and  for private use, or simply displaying nothing. 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 that is assigned to a character in the Unicode character encoding, 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, an implementation might be able to provide limited feedback to the user's queries, such as being able to sort the data properly, show its script, or otherwise display the code point in a default manner. An implementation can distinguish between unrenderable (but assigned) code points and unassigned code points by printing the former with distinctive glyphs that give some general indication of their type, such as , , , , , , , , , , and so on.

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 surro-

gate 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 meet one of only three possible conditions:

- 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 15-6* 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”).

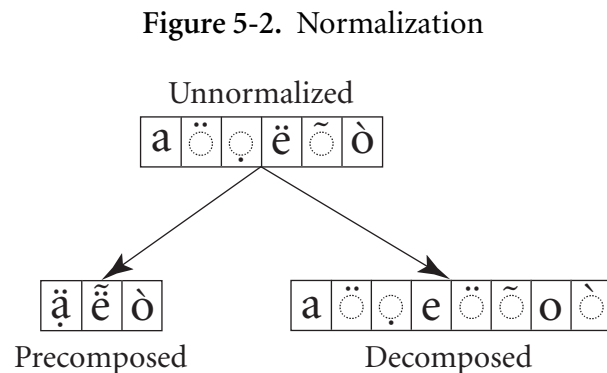
Only in higher-level protocols, such as when implementing a full mathematical formula parser, do considerations such as superscripting and subscripting of digits become crucial for numerical interpretation.

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.6, 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*).



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 Unicode Technical Report #20, “Unicode in XML and Other Markup Languages.”

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.

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 16.1, Control Codes*.

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

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

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
, 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 SimpleText 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.

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.getProperties("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

Requirements for Language Tagging

The requirement for language information embedded in plain text data is often overstated. 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.

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

Consistent Text Elements

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 alternative 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 “ı”. 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.”)

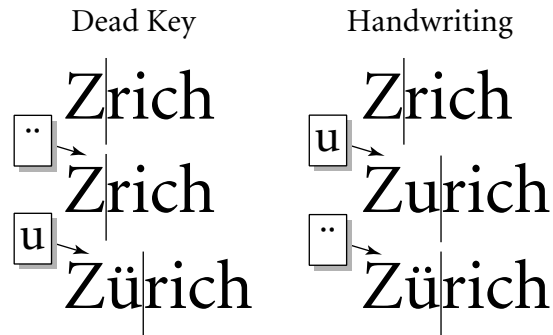
Keyboard Input

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 imple-

mented 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.

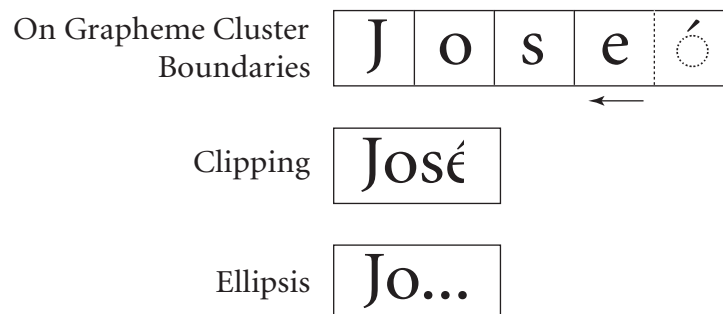
Truncation

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



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 non-

spacing 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 “ı”, 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.

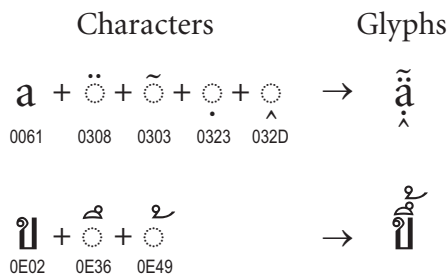
A different option is simply to clip the characters graphically. Unfortunately, this may result in clipping off part of a character, which can be visually confusing. Also, if the clipping occurs between characters, it may not give any visual feedback that characters are being omitted. A graphic or ellipsis 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

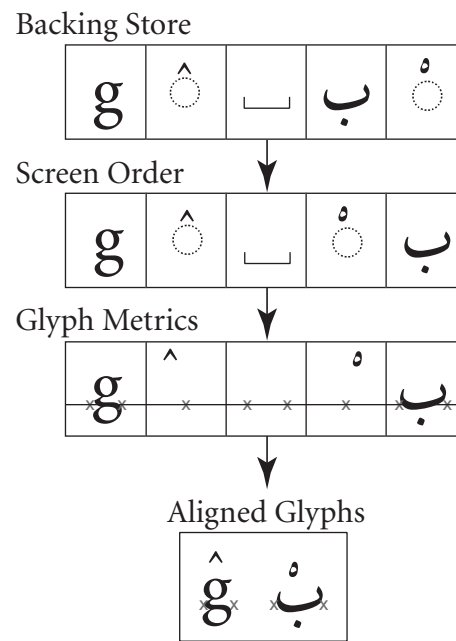


Another method (*Simple Overlap*) uses a default fixed position for an overlapping zero-width nonspacing mark. This position is generally high enough to make sure that the mark does not collide with capital letters. This will mean that this mark is placed too high above many lowercase letters. For example, 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

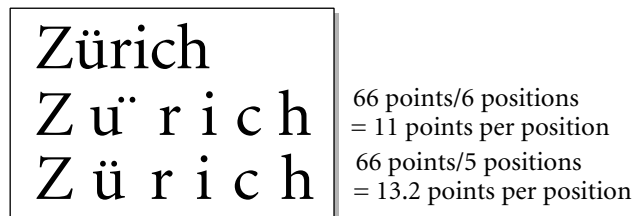


The simplest method is similar to the *Simple Overlap* fallback method. In the Bidirectional Algorithm, combining marks take the level of their base character. In that case, Arabic and Hebrew nonspacing marks would come to the left of their base characters. The font is designed so that instead of overlapping 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. Otherwise, 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



Because nonspacing marks always follow their base character, proper justification adds letterspacing between characters only if the second character is a base character.

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 ﺝ ARABIC LETTER REH + U+0651 ﺀ ARABIC SHADDA + U+064B ﺒ ARABIC FATHATAN	U+0631 ﺝ ARABIC LETTER REH + U+064B ﺒ ARABIC FATHATAN + U+0651 ﺀ 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.

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

$$\begin{aligned} a + \ddot{\circ} &\rightarrow \ddot{a} \\ A + \ddot{\circ} &\rightarrow \ddot{A} \\ f + i &\rightarrow fi \end{aligned}$$

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

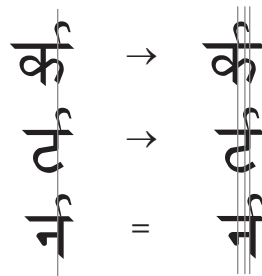
width as the base character. More sophisticated text rendering systems may take additional measures to account for those cases where the composed character sequence kerns differently or has a slightly different advance width than the base character.

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 9.1, 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



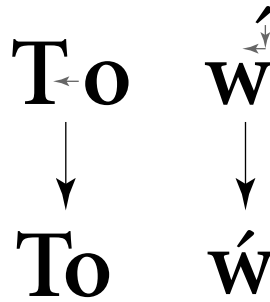
In general cases, a number of different heights of glyphs can be chosen to allow stacking of glyphs, at least for a few deep. (When these bounds are exceeded, then the fallback methods can be used.) This method can be combined with the ligature method so that in specific cases ligatures can be used to 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.

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.

Figure 5-12. Positioning with Enhanced Kerning



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 *Chapter 3, Conformance*.

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.

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*.

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 *Section 5.18, Case Mappings*.

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*, Ð *capital d-bar*, Ð *capital eth*). This approach tends to do a better job of meeting users’ expectations when searching for named files or other objects.

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 *Figure 5-13*).

Figure 5-13. Sublinear Searching

```

T h e _ q u i c k _ b r o w n ...
q u i c k
q u i c k
q u i c k
q u i c k
q u i c k
q u i c (k)

```

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 *Figure 5-1* 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.

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.

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,
    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*. For example, the archaic Georgian script contained upper- and lowercase pairs, but they are not used in modern Georgian. See *Section 7.7, Georgian*, for more information.

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), in *Section B.6, Other Unicode Online Resources*, for extensive data regarding local and language-specific casing conventions.

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.

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 ñ LATIN SMALL LETTER N PRECEDED BY APOSTROPHE. The maximum string expansion as a result of case mapping in the Unicode Standard is three. For example, uppercasing U+0390 ῖ 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 “topkapi” (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 uppercase 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 ϲ COMBINING GREEK YPOGEGRAMMENI (*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

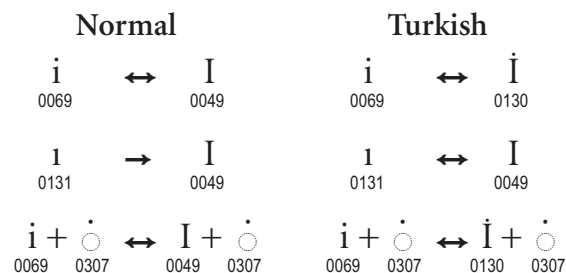
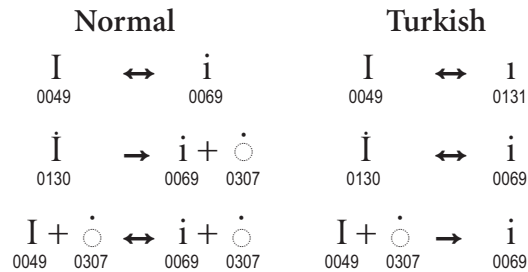


Figure 5-15 shows the lowercase mapping for Turkish *i*.

Figure 5-15. Lowercase Mapping for Turkish I

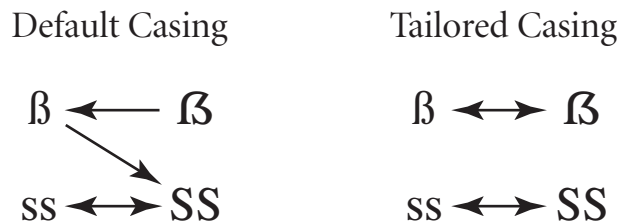


In both of the Turkish case mapping figures, a mapping with a double-sided arrow round-trips—that is, the opposite case mapping results in the original sequence. A mapping with a single-sided arrow does not round-trip.

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



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 titlecased, 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.

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 locale-independent case folding. 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.

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 characters 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, no new lowercase character will be added to the Unicode Standard as a casing pair of an existing upper- or titlecase character that has no lowercase pair. See the subsection “Policies” in *Section B.6, Other Unicode Online Resources*.

Normalization and Casing

Casing operations as defined in *Section 3.13, Default Case Algorithms*, preserve canonical equivalence, but 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)	ĵ̇	<U+01F0 LATIN SMALL LETTER J WITH CARON, U+0323 COMBINING DOT BELOW>
Uppercased	J̇̈́	<U+004A LATIN CAPITAL LETTER J, U+030C COMBINING CARON, U+0323 COMBINING DOT BELOW>
Uppercased NFC	J̇̈́	<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 $Q(X) = \text{NFC}(\text{toCasefold}(\text{NFD}(X)))$. 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 $R(X) = \text{NFKC}(\text{toCasefold}(\text{NFD}(X)))$, there are some strings such that $R(R(X)) \neq R(X)$. `FC_NFKC_Closure`, a derived property, contains the additional mappings that can be used to produce a compatibility-closed case folding. This set of mappings is 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 Unicode Technical Report #20, “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 a 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 Web site 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 lowercase 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 “MICROSOFT,” “AOL,” 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 16.2, Layout Controls* and *Section 16.4, 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.

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, but also two important format characters, U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER. The exact list of characters involved is specified by the property value: `Grapheme_Cluster_Break=Extend`.

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 means of 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.

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 16.2, Layout Controls*.

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 16.2, Layout Controls*.

Characters Ignored in Identifiers

Characters with the property `Default_Ignorable_Code_Point` (DICP) 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 `DICP=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, `DICP=True` characters are removed from a string by that transformation. With such an approach, `DICP=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.

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.

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 17.1, 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 “therSHYapist” 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 wj 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 f() FUNCTION APPLICATION has no effect on the text display and is used only in internal mathematical expression processing.

The fact that format characters and variation selectors 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.

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` (DICP). 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 `Derived-CoreProperties.txt` in the Unicode Character Database.

The `Default_Ignorable_Code_Point` property is also given to certain ranges of *unassigned* code points: U+2060..U+206F, U+FFF0..U+FFF8, and U+E0000..U+E0FFF. 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, two important sets of such format characters to note. First, there are the visible format characters which span groups of numbers, particularly for the Arabic script—for example, U+0601 ARABIC SIGN SANAH, the Arabic year sign. Such number-spanning marks always have a visible display. See “Other Signs Spanning Numbers” in *Section 8.2, Arabic* for more discussion of the use and display of these signs. The other notable set of exceptional format characters is the interlinear annotation characters: U+FFF9 INTERLINEAR ANNOTATION ANCHOR through U+FFFB INTERLINEAR ANNOTATION TERMINATION. These annotation characters should have a visible glyph display for fallback rendering, because if they are simply not displayed, there is too much potential to misread the resulting displayed text. See “Annotation Characters” in *Section 16.8, Specials* for more discussion of the use and display of interlinear annotation characters.

5.22 Best Practice for U+FFFD Substitution

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. However, there are different possible ways to use U+FFFD. This section describes the best practice.

For conversion *between* different encoding forms of the Unicode Standard, *Section 3.9, Unicode Encoding Forms* defines best practice for the use of U+FFFD. The basic formulation is as follows:

Whenever an unconvertible offset is reached during conversion of a code unit sequence:

- 1. The maximal subpart at that offset should be replaced by a single U+FFFD.*
- 2. The conversion should proceed at the offset immediately after the maximal subpart.*

In that formulation, the term “maximal subpart” refers to a *maximal subpart of an ill-formed subsequence*, which is precisely defined in *Section 3.9, Unicode Encoding Forms* for Unicode encoding forms. Essentially, a conversion algorithm gathers up the longest sequence of code units that could be the start of a valid, convertible sequence, but which is not actually convertible. For example, consider the first three bytes of a four-byte UTF-8 sequence, followed by a byte which cannot be a valid continuation byte: <F4 80 80 41>. In that case <F4 80 80> would be the maximal subpart that would be replaced by a single U+FFFD. If there is not *any* start of a valid, convertible sequence in the unconvertible data at a particular offset, then the maximal subpart would consist of a single code unit.

This practice reflects the way conversion processes are typically constructed, particularly for UTF-8. An optimized conversion algorithm simply walks an offset down the source data string until it collects a sequence it can convert or until it reaches the first offset at which it *cannot* convert that sequence. At that point it either throws an exception or it substitutes the unconvertible sequence it has collected with a single U+FFFD and then moves on to the next offset in the source.

Although the definition of best practice for U+FFFD substitution in *Section 3.9, Unicode Encoding Forms* technically applies only to conversion between Unicode encoding forms, that principle for dealing with substitution for unconvertible sequences can be extended easily to cover the more general case of conversion of any external character encoding to Unicode. The more general statement is as follows:

Whenever an unconvertible offset is reached during conversion of a code unit sequence to Unicode:

- 1. Find the longest code unit sequence that is the initial subsequence of some sequence that could be converted. If there is such a sequence, replace it with a single U+FFFD; otherwise replace a single code unit with a single U+FFFD.*
- 2. The conversion should proceed at the offset immediately after the subsequence which has been replaced.*

When dealing with conversion mappings from external character encodings to Unicode, one needs to take into account the fact that the mapping may be many-to-one. The conversion algorithm needs to find the *longest* sequence that is valid for conversion, so that it does not prematurely convert a code unit that could be part of a longer valid sequence. (This problem does not occur when converting between Unicode encoding forms, which are all constructed to be non-overlapping and one-to-one transforms.)

The requirement for finding the longest valid sequence for conversion is then generalized to the case of replacement of invalid sequences. The conversion should proceed as far as it can down the input string while the input could still be interpreted as starting *some* valid sequence. Then if the conversion fails, all of the code units that have been collected to that point are replaced with a single U+FFFD. If there is no valid code unit at all, a single code unit is replaced.

For legacy character encodings and other character encodings defined externally, the Unicode Standard cannot precisely specify what is well-formed or ill-formed. Therefore, best practice for U+FFFD substitution is defined in terms of what is convertible or unconvertible in particular cases. Ultimately, that depends on the content of character mapping tables and their accompanying conversion algorithms. To the extent that implementations share common character mapping tables, they can obtain interoperable conversion results, not only for the convertible data, but also for any data unconvertible by those tables. Unicode Technical Standard #22, “Character Mapping Markup Language,” provides an XML format for precisely specifying character mapping tables, which can be used to help guarantee interoperable conversions.

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*. Character blocks generally contain characters from a single script, and in many cases, a script is fully represented in its character 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 character blocks. Corresponding subsection headers identify each block and its associated range of Unicode code points. The Unicode code charts are also organized by character 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 character blocks, including Basic Latin, Latin-1 Supplement, General Punctuation, Supplemental Punctuation, and CJK Symbols and Punctuation. There are also occasional punctuation characters in character 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 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 9.1, 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

$$\begin{array}{ll} \text{ka} + \text{i} \rightarrow \text{ki} & \text{ka} + \text{e} \rightarrow \text{ke} \\ \text{ka} + \text{u} \rightarrow \text{ku} & \text{ka} + \text{o} \rightarrow \text{ko} \end{array}$$

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.

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, encodes an explicit virama character and represents text in its logical order. The Thai model departs from the Devanagari model in that it represents text in its visual display order, based on the typewriter legacy, rather than in logical order. The Tibetan model avoids an explicit virama, instead encoding a sequence of *subjoined consonants* to represent consonants occurring in clusters in a syllable.

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, many of which may eventually be encoded in the Unicode Standard. 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 14.18, Egyptian Hieroglyphs*.

The classification of a writing system is often somewhat blurred 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}. 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*).

Table 6-1. Typology of Scripts in the Unicode Standard

Alphabets	Latin, Greek, Cyrillic, Armenian, Thaana, Mandaic, Georgian, Ogham, Runic, Mongolian, Glagolitic, Coptic, Tifinagh, Old Italic, Gothic, Ugaritic, Old Persian, Deseret, Shavian, Osmanya, N’Ko, Ol Chiki, Kayah Li, Carian, Lycian, Lydian, Avestan, Lisu, Old Turkic, Meroitic Cursive, Meroitic Hieroglyphs
Abjads	Hebrew, Arabic, Syriac, Phoenician, Samaritan, Imperial Aramaic, Old South Arabian, Inscriptional Parthian, Inscriptional Pahlavi
Abugidas	Devanagari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Sinhala, Thai, Lao, Tibetan, Myanmar, Tagalog, Hanunóo, Buhid, Tagbanwa, Khmer, Limbu, Tai Le, New Tai Lue, Buginese, Syloti Nagri, Kharoshthi, Balinese, Phags-pa, Sundanese, Batak, Lepcha, Saurashtra, Rejang, Cham, Tai Tham, Tai Viet, Javanese, Meetei Mayek, Brahmi, Kaithi, Chakma, Sharada, Sora Sompeng, Takri
Logosyllabaries	Han, Sumero-Akkadian, Egyptian Hieroglyphs
Simple Syllabaries	Cherokee, Hiragana, Katakana, Bopomofo, Yi, Linear B, Cypriot, Ethiopic, Canadian Aboriginal Syllabics, Vai, Bamum, Miao
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, 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.

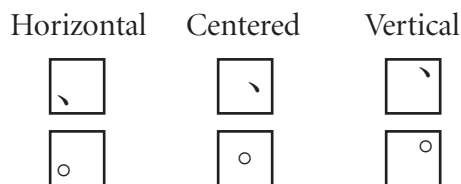
Rendering. Punctuation characters vary in appearance with the font style, just like the surrounding text characters. In some cases, where used in the context of a particular script, a specific glyph style is preferred. For example, U+002E FULL STOP should appear square when used with Armenian, but is 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



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 16.2, Layout Controls*.

Encoding Characters with Multiple Semantic Values. Some of the punctuation characters in the ASCII 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, 27₁₆ is defined in ANSI X3.4 as *apostrophe (closing single quotation mark; acute accent)*, and 2D₁₆ 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.

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.

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 SEPARATOR, U+2029 PARAGRAPH SEPARATOR, and U+2060 WORD JOINER, are described in *Section 16.2, Layout Controls*. Bidirectional ordering controls are also discussed in *Section 16.2, Layout*

Controls, but their detailed use is specified in Unicode Standard Annex #9, “Unicode Bidirectional Algorithm.”

Invisible operators are explained in *Section 15.6, Invisible Mathematical Operators*. Deprecated format characters related to obsolete models of Arabic text processing are described in *Section 16.3, Deprecated Format Characters*.

The reserved code points U+2064..U+2069 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*.

Space Characters

Space characters are found in several character 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+180E	MONGOLIAN VOWEL SEPARATOR
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 16.2, 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 non-breaking 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, which except for its display width behaves exactly the same in its line breaking behavior. It is regularly used in Mongolian in certain grammatical contexts (before a particle), where it also influences the shaping of the glyphs for the particle. In Mongolian text, the NNBSP is typically displayed with 1/3 the width of a normal space character. The NNBSP can be used to represent the narrow space occurring around punctuation characters in French typography, which is called an “espace fine insécable.”

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 be used to

indicate a *binary minus sign*. U+2015 HORIZONTAL BAR is used to introduce quoted text in some typographic styles.

Dashes and hyphen characters may also be found in other character blocks in the Unicode Standard. A list of dash and hyphen characters appears in *Table 6-3*. For a description of the line breaking behavior of dashes and hyphens, see Unicode Standard Annex #14, “Unicode Line Breaking Algorithm.”

Table 6-3. Unicode Dash Characters

Code	Name
U+002D	HYPHEN-MINUS
U+007E	TILDE (when used as <i>swung dash</i>)
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 (= <i>quotation dash</i>)
U+2053	SWUNG DASH
U+207B	SUPERSCRIPIT MINUS
U+208B	SUBSCRIPT MINUS
U+2212	MINUS SIGN
U+2E17	DOUBLE 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

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 16.2, 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 Diacritics” in *Section 7.1, Latin*). This is a form common in older implementations, particularly for terminal emulation and type-writer-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.

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.

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. Unlike brackets, quotation marks are not mirrored in a bidirectional context.

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 15.7, Technical Symbols*, for more information.

Language-Based Usage of Quotation Marks

U+0022 QUOTATION MARK is the most commonly used character for quotation mark. However, it has ambiguous semantics and direction. Most keyboard layouts support only U+0022 QUOTATION MARK, therefore word processors commonly offer a facility for automatically converting the U+0022 QUOTATION MARK to a contextually selected curly quote glyph.

European Usage. The use of quotation marks differs systematically by language and by medium. 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. 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»

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 6-4*.

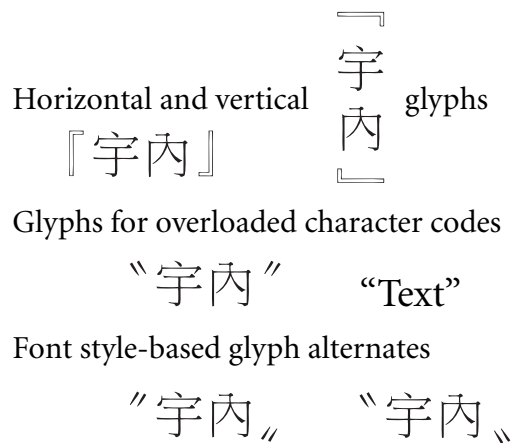
Glyph Variation. The glyphs for “double-prime” quotation marks consist of a pair of wedges, slanted either forward or backward, with the tips of the wedges pointing either up

Table 6-4. East Asian Quotation Marks

Style	Opening	Closing
Corner bracket	300C	300D
White corner bracket	300E	300F
Double prime	301D	301F

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



Three pairs of quotation marks are used with Western-style horizontal text, as shown in *Table 6-5*.

Table 6-5. 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, U+00BB (double guillemets), and U+201D RIGHT DOUBLE QUOTATION MARK are context dependent. The semantics of U+201A and U+201B LOW-9 QUOTATION MARKS are always opening; this usage is distinct from the usage 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.

Apostrophes

U+0027 APOSTROPHE 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 RIGHT SINGLE QUOTATION MARK is preferred as apostrophe, but only U+0027 is present on keyboards. Word processors commonly offer a facility for automatically converting the U+0027 APOSTROPHE 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 MODIFIER LETTER APOSTROPHE 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 RIGHT SINGLE QUOTATION MARK 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.

Other Punctuation

Hyphenation Point. U+2027 HYPHENATION POINT is a raised dot used to indicate correct word breaking, as in dic-tion-ar-ies. It is a punctuation mark, to be distinguished from U+00B7 MIDDLE DOT, which has multiple semantics.

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 $\frac{3}{4}$ 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. For other characters in this range that have alternative glyphs, the Unicode character is displayed with the basic or most common glyph; rendering software may present any other graphical form of that character.

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. For example, in a monowidth font, a sequence of three *full stops* will be wider than the *horizontal ellipsis*, but in a typical proportional font, a *full stop* is very narrow and a sequence of three of them will be more tightly spaced than the the dots in *horizontal ellipsis*.

Conventions that use four dots for an ellipsis in certain grammatical contexts should represent them 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.

In East Asian typographic traditions, particularly in Japan, an ellipsis is raised to the center line of text. This effect requires the use of a Japanese-specific font, or at least a specific glyph for the *horizontal ellipsis* character.

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 MIDLIN HORIZONTAL ELLIPSIS are part of a set of special ellipsis characters used for row or column elision in matrix notation. Their use is restricted to mathematical contexts; they should not be used as glyph variants of the ordinary punctuation ellipsis for East Asian typography.

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.

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.

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 6-6 gives a sample.

Table 6-6. Names for the @

Language	Name and Comments
Chinese	= xiao laoshu (means “little mouse” in Mandarin Chinese), laoshu hao (means “mouse mark” in Mandarin Chinese)
Danish	= grishale, snabel-a (common, humorous slang)
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)

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.

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 SIDEWAYS 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, *stigma*, *hypodiatole*). The *hypodiatole* 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 *downwards 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*.

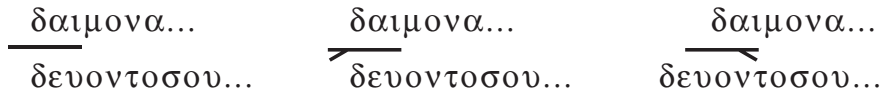
Figure 6-5. Examples of Ancient Greek Editorial Marks

5	↗ αιθει . [ταναίο[όμπαυ[υ[·]κα[$\frac{\text{⋮}}{\text{⋮}}$ τίστ' ώπου[⊕ ού μέν πως ... ⊕ ούκ άγαθόν πολ < εἷς βασιλεύς, ᾧ ... ∕ σκῆπτρόν τ' ἦδ' ...
10	εἶπη[παρεσκεθ' ὀ[δάμιον' ανάιτιο[----- δεινοντοσουδεπ[⊖ αὔριον ἦν ἀρετήν ... ⊖ μείνη ἐπερχόμενον ...

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 para-

graph 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



Double Oblique Hyphen. U+2E17 “*z*” 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.

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 6-7.

Table 6-7. 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+1B5E	BALINESE CARIK SIKI
U+1B5F	BALINESE CARIK PAREREN
U+1C3B	LEPCHA PUNCTUATION TA-ROL
U+1C3C	LEPCHA PUNCTUATION NYET THYOOM TA-ROL

Table 6-7. Unicode Danda Characters (Continued)

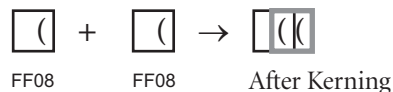
Code	Name
U+1C7E	OL CHIKI PUNCTUATION MUCAAD
U+1C7F	OL CHIKI PUNCTUATION DOUBLE MUCAAD
U+A876	PHAGS-PA SHAD
U+A877	PHAGS-PA MARK DOUBLE SHAD
U+A8CE	SAURASHTRA DANDA
U+A8CF	SAURASHTRA DOUBLE DANDA
U+A92F	KAYAH LI SIGN SHYA
U+AA5D	CHAM PUNCTUATION DANDA
U+AA5E	CHAM PUNCTUATION DOUBLE DANDA
U+AA5F	CHAM PUNCTUATION TRIPLE DANDA
U+AAF0	MEETEI MAYEK CHEIKHAM
U+ABEB	MEETEI MAYEK CHEIKHEI
U+10A56	KHAROSHTHI PUNCTUATION DANDA
U+10A57	KHAROSHTHI PUNCTUATION DOUBLE DANDA
U+11141	CHAKMA DANDA
U+11142	CHAKMA DOUBLE DANDA
U+111C5	SHARADA DANDA
U+111C6	SHARADA 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.

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 character space adjustment of wide punctuation characters is based on their character code.

Figure 6-7. CJK Parentheses

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+301C WAVE DASH and U+3030 WAVY DASH are special forms of dashes 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.

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*.

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 IDEO-

GRAPHIC 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 12.1, Han*.

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 12.4, 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: 2131₁₆ was unified with U+00B7 MIDDLE DOT, and 2261₁₆ 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 12.5, Halfwidth and Fullwidth Forms*, for more information.

Chapter 7

European Alphabetic 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.

The alphabetic scripts and additional characters described in this chapter are:

<i>Latin</i>	<i>Cyrillic</i>	<i>Georgian</i>
<i>Greek</i>	<i>Glagolitic</i>	<i>Modifier letters</i>
<i>Coptic</i>	<i>Armenian</i>	<i>Combining marks</i>

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.

The Latin alphabet is derived from the alphabet used by the Etruscans, who had adopted a Western variant of the classical Greek alphabet (*Section 14.2, 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. Modern Georgian does not have separate uppercase and lowercase forms.

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 14, Additional Ancient and Historic Scripts*.

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 interoperability with pervasive practice in legacy encodings. Combining diacritical marks can express these and all other accented letters as combining character sequences.

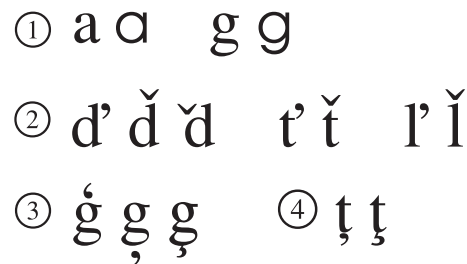
In the Unicode Standard, all diacritical marks are encoded in sequence *after the base characters to which they apply*. For more details, see the subsection “Combining Diacritical Marks” in *Section 7.9, Combining Marks*, and also *Section 2.11, Combining Characters*.

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.

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 and Pinyin, 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

Figure 7-1. Alternative Glyphs in Latin



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. There are two notable special cases regarding the use of these diacritic marks below letters which require further discussion: Latvian cedillas and the use of cedillas or comma below in Turkish and Romanian.

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*, 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 § LATIN SMALL LETTER S WITH COMMA BELOW versus U+015F § LATIN SMALL LETTER S WITH CEDILLA. In modern usage, the preferred representation of Romanian text is with U+0219 § LATIN SMALL LETTER S WITH COMMA BELOW, while Turkish data is represented with U+015F § LATIN SMALL LETTER S WITH CEDILLA.

However, due to the prevalence of legacy implementations, a large amount of Romanian data will contain U+015F ș 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 ș 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 a nonspacing mark above loses the dot in rendering. Thus, in the word *naïve*, the *i* could be spelled with *i* + *diaeresis*. A *dotless-i* is not equivalent to a Turkish *dotless-i* + *overdot*, nor are other cases of accented *dotless-i* equivalent to accented *dotless-i* (for example, $i + \grave{\cdot} \neq i + \grave{\cdot}$). 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*

$$\begin{array}{l} \mathring{i} + \ddot{\cdot} \rightarrow \ddot{i} \quad \mathring{i} + \overset{\circ}{\cdot} + \overset{\circ}{\cdot} \rightarrow \overset{\circ}{\overset{\circ}{i}} \\ \mathring{j} + \overset{\circ}{\cdot} \rightarrow \overset{\circ}{j} \quad \mathring{i} + \overset{\circ}{\cdot} + \overset{\circ}{\cdot} \rightarrow \overset{\circ}{\overset{\circ}{i}} \end{array}$$

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 Ă 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

a ă â e ê i o ô ơ u ư y
 ̀ ̂ ̃ ̄ ̅ ̆

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 15, 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).

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”.

Letters of the Latin-1 Supplement: U+00C0–U+00FF

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.

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.

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 Latin 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 ’ MODIFIER LETTER APOSTROPHE, U+02BB ‘ MODIFIER LETTER TURNED COMMA, U+02C0 ʔ MODIFIER LETTER GLOTTAL STOP, or U+02BE ’ MODIFIER LETTER RIGHT HALF RING. U+02BB, in particular, is used in Hawaiian orthography as the *’okina*.

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 last published IPA revision (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. Note also that a few nonstandard or obsolete phonetic symbols are encoded in the Latin Extended-B block.

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 LATIN SMALL LETTER IOTA “ı” versus U+026A LATIN LETTER SMALL CAPITAL I “ı̇.” 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.

Phonetic Extensions: U+1D00–U+1DBF

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 *thin*. 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.

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 uppcased 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 ngữ*) combined with a diacritic mark that denotes the phonemic tone that applies to the syllable.

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.

Uighur. The Latin orthography for the Uighur 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 Uighur.

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.

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.

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 range U+A730..U+A778 occur in a variety of European medievalist manuscript traditions. None of these have any modern orthographic usage. A number of these letter forms 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.

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 diacritic 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.

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 “Controlling Ligatures” in *Section 16.2, Layout Controls*.

7.2 Greek

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 7-1*).

Table 7-1. Nonspacing Marks Used with Greek

Code	Name	Alternative Names
U+0300	COMBINING GRAVE ACCENT	<i>varia</i>
U+0301	COMBINING ACUTE ACCENT	<i>tonos, oxia</i>
U+0304	COMBINING MACRON	
U+0306	COMBINING BREVE	
U+0308	COMBINING DIAERESIS	<i>dialytika</i>
U+0313	COMBINING COMMA ABOVE	<i>psili, smooth breathing mark</i>
U+0314	COMBINING REVERSED COMMA ABOVE	<i>dasia, rough breathing mark</i>
U+0342	COMBINING GREEK PERISPOMENI	<i>circumflex, tilde, inverted breve</i>
U+0343	COMBINING GREEK KORONIS	<i>comma above</i>
U+0345	COMBINING GREEK YPOGEGRAMMENI	<i>iota subscript</i>

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 $\hat{\sigma}$, an inverted breve $\sigmâ$, a tilde $\tilde{\sigma}$, or occasionally a macron $\bar{\sigma}$. 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 titlecase 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 uppercase 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 *proseegrammeni* (also known as *iota adscript* in English). In completely uppercased 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

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 15.5, Mathematical Symbols*. For additional punctuation to be used with this script, see C0 Controls and ASCII Punctuation (U+0000..U+007F).

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 diacritic 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 7-2*. The spacing forms are meant for keyboards and pedagogical use and are not to be used in the representation of titlecase words. The compatibility decompositions of these spacing forms consist of the sequence U+0020 SPACE followed by the nonspacing form equivalents shown in *Table 7-2*.

Table 7-2. Greek Spacing and Nonspacing Pairs

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

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

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 Coptologists 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 (*ie*, *ni*, and *fi*) were substituted by themselves. However, because *ie* 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 Ⲭ 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 Ⲭ 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 COM-

BINING GRAVE ACCENT. Other dialects, including Sahidic, use U+0305 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
ⲁ̅ or ⲁ̅̅	1,000
ⲁ̅̅̅̅	1,888

U+0374 GREEK NUMERAL SIGN is used to indicate fractions. For example, ρ 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. In the nineteenth and early twentieth centuries, Cyrillic was extended to write the non-Slavic minority languages of Russia and neighboring countries.

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 “І” CYRILLIC CAPITAL LETTER BYELORUSSIAN-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*.

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 “I” 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 “I” CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I; however, in some serified 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.

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 *љ* and *њ*, which are ligatures of the base letters *л* and *н* with a pal-

atalizing soft sign *ь*. The Molodtsov orthography made use of a different kind of palatalization hook for Komi *ӧ, ӧ, ӧ, ӧ*, 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.

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, usually to indicate abbreviations of words in the text. These may occur with or without the generic titlo character, U+0483 COMBINING CYRILLIC TITLO, or with U+A66F COMBINING CYRILLIC VZMET.

The glyphs in the code charts are based on the modern Cyrillic letters to which these letter titlos correspond, but in Old Church Slavonic manuscripts, the actual glyphs used are related to the older forms of Cyrillic letters.

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.

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.

7.5 Glagolitic

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.

7.6 Armenian

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 Maštoc⁴ 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 *verjakēt* 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 *mijakēt*; 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 it is possible to differentiate between word-joining and word-splitting hyphens. To join words, the *miowtʻjan 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 *entʻamna* 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

Georgian: U+10A0–U+10FF, 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 now the form used for nearly all modern Georgian writing.

Both the *Mkhedruli* alphabet and the *Asomtavruli* inscriptional form are encoded in the Georgian block. The *Nuskhuri* script form is encoded in the Georgian Supplement block.

Case Forms. The Georgian *Mkhedruli* alphabet is fundamentally caseless. 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*. This practice is anomalous—the Unicode Standard instead provides case mappings between the two *Khutsuri* forms: *Asomtavruli* and *Nuskhuri*.

Mtavruli Style. *Mtavruli* is a particular style of *Mkhedruli* in which the distinction between letters with ascenders and descenders is not maintained. All letters appear with an equal height standing on the baseline; *Mtavruli*-style letters are never used as capitals. A word is always entirely presented in *Mtavruli* or not. *Mtavruli* is a font style, similar to SMALL CAPS in the Latin script.

Figure 7-6 illustrates the various forms of Georgian and its case usage discussed in the text, using Akaki Shanidze’s name.

Figure 7-6. Georgian Scripts and Casing

Asomtavruli majuscule	ԱԿԱԿԻ ՄԵՄԻԺԻ
Nuskhuri minuscule	ակակի მემიძი
Casing Khutsuri	ԱԿԱԿԻ ՄԵՄԻԺԻ
Mkhedruli	აკაკი მანძი
Mtavruli style	აკაკი მანძი
Shanidze’s orthography	Աკაკი მანძი

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 *ʻokina* (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 modifiers 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 nevertheless 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.

Names. There is no requirement that the Unicode names for modifier letters contain the label “MODIFIER LETTER”, although most of them do.

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 covered in the Basic Latin and Latin-1 Supplement blocks. The six common European diacritics that do not have encodings there are encoded as spacing characters. 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 free-standing 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 7-7*).

Figure 7-7. Tone Letters

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 7-7* 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 7-7*, to indicate a low rising-falling contour tone.

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 “ä”, *not* “äu”, 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 Overlaid Diacritics” in *Section 2.12, Equivalent Sequences and Normalization* 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. 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 7-8*.

Figure 7-8. Double Diacritics

$$\begin{array}{ccc}
 \mathbf{n} + \overset{\circ}{\sim} & \rightarrow & \overset{\circ}{\sim}\mathbf{n} \\
 \text{006E} \quad \text{0360} & & \\
 \mathbf{n} + \overset{\circ}{\sim} + \mathbf{g} & \rightarrow & \overset{\circ}{\sim}\mathbf{ng} \\
 \text{006E} \quad \text{0360} \quad \text{0067} & &
 \end{array}$$

These double-diacritic marks have a very high combining class—higher than all other non-spacing 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 7-9*.

Figure 7-9. Positioning of Double Diacritics

$$\begin{array}{ccc}
 \mathbf{a} + \overset{\circ}{\hat{}} + \overset{\circ}{\sim} + \mathbf{c} + \overset{\circ}{\ddot{}} & \rightarrow & \overset{\circ}{\hat{}}\overset{\circ}{\sim}\mathbf{ac} \\
 \text{0061} \quad \text{0302} \quad \text{0360} \quad \text{0063} \quad \text{0308} & & \\
 \mathbf{a} + \overset{\circ}{\sim} + \overset{\circ}{\hat{}} + \mathbf{c} + \overset{\circ}{\ddot{}} & \rightarrow & \overset{\circ}{\sim}\overset{\circ}{\hat{}}\mathbf{ac} \\
 \text{0061} \quad \text{0360} \quad \text{0302} \quad \text{0063} \quad \text{0308} & &
 \end{array}$$

In *Figure 7-9*, 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 7-10*.

Figure 7-10. Use of CGJ with Double Diacritics

$$\begin{array}{ccccccccc} \mathbf{u} & + & \text{◌} & + & \text{◌} & + & \mathbf{i} & \rightarrow & \mathbf{ui} \\ \text{0075} & & \text{0361} & & \text{034F} & & \text{0301} & & \text{0069} \end{array}$$

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 16.2, Layout Controls*.

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 7-11 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 7-11*. Note that the two characters *f* and *i* may not ligate, even if the sequence <*f*, *i*> does.

Figure 7-11. Interaction of Combining Marks with Ligatures

$$\begin{array}{l} \textcircled{1} \mathbf{f} + \mathbf{i} + \text{◌} \hat{\text{◌}} \equiv \mathbf{f} + \mathbf{\hat{i}} \rightarrow \mathbf{fi}, \mathbf{\hat{fi}}, \mathbf{\hat{f}i} \\ \textcircled{2} \mathbf{f} + \text{◌} \tilde{\text{◌}} + \mathbf{i} + \text{◌} \hat{\text{◌}} \rightarrow \mathbf{\tilde{f}\hat{i}}, \mathbf{\tilde{f}i} \\ \textcircled{3} \mathbf{f} + \text{◌} \hat{\text{◌}} + \mathbf{i} + \text{◌} \tilde{\text{◌}} \rightarrow \mathbf{\hat{f}\tilde{i}}, \mathbf{\hat{f}i} \\ \textcircled{4} \mathbf{f} + \text{◌} \tilde{\text{◌}} + \mathbf{i} + \text{◌} \hat{\text{◌}} \not\equiv \mathbf{f} + \text{◌} \hat{\text{◌}} + \mathbf{i} + \text{◌} \tilde{\text{◌}} \end{array}$$

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.

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).

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.

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.

Combining 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 7-12. 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 7-12. Use of Vertical Line Overlay for Negation



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 3.11, Normalization Forms*.

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.

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 letterforms. 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 discontinuous sequence of the combining half marks corresponds to a single combining mark, as depicted in *Figure 7-13*. The preferred forms are the double diacritics, such as U+0360 COMBINING DOUBLE TILDE.

This block also contains two half macron marks and a conjoining macron mark. These combining marks are intended for use to support a particular style of supralineation in Coptic. See *Section 7.3, Coptic*.

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 vowelings of Arabic text as well as another collection of combining marks used in annotation of Koranic 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*.

Figure 7-13. Double Diacritics and Half Marks

Using Combining Half Marks

$$\underset{006E}{\mathbf{n}} + \underset{FE22}{\overset{\circ}{\sim}} + \underset{0067}{\mathbf{g}} + \underset{FE23}{\overset{\circ}{\sim}} \rightarrow \overset{\circ}{\sim}\mathbf{ng}$$

Using Double Diacritics

$$\underset{006E}{\mathbf{n}} + \underset{0360}{\overset{\sim}{\sim}} + \underset{0067}{\mathbf{g}} \rightarrow \overset{\sim}{\sim}\mathbf{ng}$$

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

Middle Eastern Scripts

The scripts in this chapter have a common origin in the ancient Phoenician alphabet. They include:

<i>Hebrew</i>	<i>Samaritan</i>
<i>Arabic</i>	<i>Thaana</i>
<i>Syriac</i>	

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 Thaana script is used to write Dhivehi, the language of the Republic of Maldives, an island nation in the middle of the Indian Ocean.

The Middle Eastern scripts are mostly abjads, with small character sets. Words are demarcated by spaces. Except for Thaana, 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 and Syriac are cursive scripts even when typeset, unlike Hebrew, Samaritan, and Thaana, where letters are unconnected. Most letters in Arabic and Syriac assume different forms depending on their position in a word. Shaping rules for the rendering of text are specified in *Section 8.2, Arabic*, and *Section 8.3, Syriac*. 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 marks of pronunciation (“vocalization”) are encoded as combining characters, so support for vocalized text necessitates use of composed character sequences. Yiddish, Syriac, and Thaana are normally written with vocalization; Hebrew, Samaritan, and Arabic are usually written unvocalized.

8.1 Hebrew

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 Marks of Pronunciation. 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 marks of punctuation.

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 *marks of pronunciation*: 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 letterforms 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 16.2, *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.

Currency Symbol. The NEW SHEQEL SIGN (U+20AA) is encoded in the currency block.

Alphabetic Presentation Forms: U+FB1D–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.

8.2 Arabic

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 Kurdish, 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.

The Arabic script is cursive, even in its printed form (see *Figure 8-1*). 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 *harakat*, which are applied to consonantal base letters. In normal writing, however, these *harakat* are omitted.

Figure 8-1. Directionality and Cursive Connection

Memory representation:	٥٥٥ ٥
After reordering:	٥ ٥٥٥
After joining:	٥ ٥٥٥

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:** Diacritic 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* diacritic 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), and *shaddah* (consonant gemination mark). 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. It occurs only above *alef*, and in that combination represents the sound /ʔaa/. For historical reasons, the precomposed combination U+0622 ARABIC LETTER ALEF WITH MADDAH ABOVE is encoded, but 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*.
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 in text, 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:** Koranic 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. 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.

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 8-2*.

Figure 8-2. Using a Joiner

Memory representation: ٥٥٥ ZW ٥
 After reordering: ٥ ZW ٥٥٥
 After joining: ٥ ٤ ٥

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 8-3*. Examples include the Persian plural suffix, some Persian proper names, and Ottoman Turkish vowels.

Figure 8-3. Using a Non-joiner

Memory representation: ٥ ZW NJ ٥٥ ٥
 After reordering: ٥ ٥٥ ZW NJ ٥
 After joining: ٥ ٤ ٥٥

Joiners and non-joiners may also occur in combinations. The effects of such combinations are shown in *Figure 8-4*. For further discussion of joiners and non-joiners, see *Section 16.2, Layout Controls*.

Figure 8-4. Combinations of Joiners and Non-joiners

Memory representation: ٥ ZW NJ ZW J ٥٥ ٥
 After reordering: ٥ ٥٥ ZW J ZW NJ ٥
 After joining: ٥ ٤ ٥٥

Harakat (Vowel) Nonspacing Marks. *Harakat* are marks that indicate vowels or other modifications of consonant letters. The code charts depict a character in the harakat range in relation to a dashed 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. The Unicode Standard does not specify a sequence order in case of multiple harakat applied to the same Arabic base character, as there is no possible ambiguity of interpretation. For more information about the canonical ordering of nonspacing marks, see *Section 2.11, Combining Characters*, and *Section 3.11, Normalization Forms*.

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+064D ڤ ARABIC KASRATAN the glyph is placed below the base character, as shown in the first example in *Figure 8-5*. However, computer fonts often follow an approach that originated in metal typesetting and combine the *kasratan* with *shadda* in a ligature placed above the text, as shown in the second example in *Figure 8-5*.

Figure 8-5. Placement of Harakat



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 ·١٢٣...). 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 Iran, Pakistan, and Afghanistan (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 *Table 8-1*. A different set of digits, called Rumi, was used in historical materials from Egypt to Spain, and is discussed in the subsection on “Rumi Numeral Forms” in *Section 15.3, Numerals*.

There is substantial variation in usage of glyphs for the Eastern Arabic-Indic digits, especially for the digits four, five, six, and seven. *Table 8-2* illustrates this variation with some example glyphs for digits in languages of Iran, Pakistan, and India. While some usage of the

Table 8-1. 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

Persian glyph for U+06F7 EXTENDED ARABIC-INDIC DIGIT SEVEN can be documented for Sindhi, the form shown in Table 8-2 is predominant.

Table 8-2. Glyph Variation in Eastern Arabic-Indic Digits

Code Point	Digit	Persian	Sindhi	Urdu
U+06F4	4	۴	۴	۴
U+06F5	5	۵	۵	۵
U+06F6	6	۶	۶	۶
U+06F7	7	۷	۷	۷

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. (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 and the Arabic Extended-A block, U+08A0..U+08FF.

Koranic Annotation Signs. These characters are used in the Koran to mark pronunciation and other annotation. The enclosing mark U+06DE is used to enclose a digit. When rendered, the digit appears in a smaller size. Several additional Koranic annotation signs are encoded in the Arabic Extended-A block, U+08A0..U+08FF.

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. Depending on the letter shapes present in the name and the calligraphic style in use, the honorific mark may be applied to a letter somewhere in the middle of the name. The normalization algorithm does not move such word-level combining characters to the end of the word.

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+FDFA RIAL SIGN is a currency symbol used in Iran. Unlike the AFGHANI SIGN, U+FDFA 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>.

End of Ayah. U+06DD ARABIC END OF AYAH graphically encloses a sequence of zero or more digits (of General Category Nd) that follow it in the data stream. The enclosure terminates with any non-digit. For behavior of a similar prefixed formatting control, see the discussion of U+070F SYRIAC ABBREVIATION MARK in *Section 8.3, Syriac*.

Other Signs Spanning Numbers. Several other special signs are written in association with numbers in the Arabic script. U+0600 ARABIC NUMBER SIGN signals the beginning of a number; it is written below the digits of the number.

U+0601 ARABIC SIGN SANAH indicates a year (that is, as part of a date). This sign is 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. The use of the year sign is illustrated in *Figure 8-6*.

Figure 8-6. Arabic Year Sign



U+0602 ARABIC FOOTNOTE MARKER is another of these signs; it is used in the Arabic script in conjunction with the footnote number itself. It also precedes the digits in logical order and is written to extend underneath them.

Finally, U+0603 ARABIC SIGN SAFHA functions as a page sign, preceding and extending under a sequence of digits for a page number.

Like U+06DD ARABIC END OF AYAH, 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 the sequence of digits that follows them. They *precede* the sequence of digits that they span, rather than following a base character, as would be the case for a combining mark. Their General Category value is Cf (format control character). Unlike most other format control 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.

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.

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, 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 8-3*. (See *ArabicShaping.txt* in the Unicode Character Database for a complete list.) In this table, *right* and *left* refer to visual order. The characters of the right-joining type are exemplified in more detail in *Table 8-9*, and those of the dual-joining type are shown in *Table 8-8*. When characters do not join or cause joining (such as DAMMATAN), they are classified as transparent.

Table 8-3. Primary Arabic Joining Types

Description	Joining Type	Examples and Comments
Right-joining	R	ALEF, DAL, THAL, REH, ZAIN ...
Left-joining	L	None
Dual-joining	D	BEH, TEH, THEH, JEEM ...
Join-causing	C	U+200D ZERO WIDTH JOINER and TATWEEL (0640). 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 HAMZA (0621), HIGH HAMZA (0674), spaces, digits, punctuation, non-Arabic letters, and so on. Also, U+0600 ARABIC NUMBER SIGN, U+0603 ARABIC SIGN SAFHA 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 FATHATAN (064B) and other Arabic <i>harakat</i> , HAMZA BELOW (0655), SUPERSCRIPT ALEF (0670), combining Koranic annotation signs, and nonspacing marks from other scripts. Also U+070F SYRIAC ABBREVIATION MARK.

Table 8-4 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 8-4. 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 8-5.

Table 8-5. Arabic Glyph Types

Glyph Type	Description
X_n	Nominal glyph form as it appears in the code charts
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:*

$$\text{MEEM}_n + \text{SHADDA}_n + \text{LAM}_n \rightarrow \text{MEEM}_r + \text{SHADDA}_n + \text{LAM}_l$$



- R2** *A right-joining character X that has a right join-causing character on the right will adopt the form X_r. For example:*

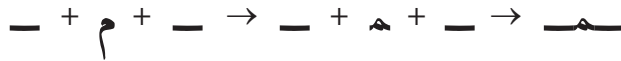
$$\text{ALEF}_n + \text{TATWEEL}_n \rightarrow \text{ALEF}_r + \text{TATWEEL}_n$$



- 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:*

$$\text{TATWEEL}_n + \text{MEEM}_n + \text{TATWEEL}_n \rightarrow \text{TATWEEL}_n + \text{MEEM}_m + \text{TATWEEL}_n$$



- 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:*

$$\text{MEEM}_n + \text{TATWEEL}_n \rightarrow \text{MEEM}_r + \text{TATWEEL}_n$$



- 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:*

$$\text{TATWEEL}_n + \text{MEEM}_n \rightarrow \text{TATWEEL}_n + \text{MEEM}_l$$



- R7** *If none of the preceding rules applies to a character X, then it will adopt the nominal 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.

Arabic Ligatures

Ligature Classes. Certain types of ligatures are obligatory in Arabic script regardless of font design. Many other optional ligatures are possible, depending on font design. Because they are optional, those ligatures are not covered in this discussion.

For the purpose of describing the obligatory Arabic ligatures, certain characters fall into two joining groups, as shown in *Table 8-6*. The complete list is available in ArabicShaping.txt in the Unicode Character Database.

Table 8-6. Arabic Obligatory Ligature Joining Groups

Joining Group	Examples
ALEF	MADDA-ON-ALEF, HAMZA ON ALEF, ...
LAM	LAM, LAM WITH SMALL V, LAM WITH DOT ABOVE, ...

Ligature Rules. The following rules describe the formation of 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 8-7.

Table 8-7. Arabic Ligature Notation

Symbol	Description
$(X-Y)_n$	Nominal ligature glyph form representing a combination of an X_r form and a Y_l form
$(X-Y)_r$	Right-joining ligature glyph form representing a combination of an X_r form and a Y_m form
$(X-Y)_l$	Left-joining ligature glyph form representing a combination of an X_m form and a Y_l form
$(X-Y)_m$	Dual-joining (medial) ligature glyph form representing a combination of an X_m 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 $(\text{LAM-ALEF})_r$. For example:

$$\text{ا} + \text{ل} \rightarrow \text{لا} \quad (\text{not } \text{لا})$$

L3 Any sequence with ALEF_r on the left and LAM_l on the right will form the ligature $(\text{LAM-ALEF})_n$. For example:

$$\text{ا} + \text{ل} \rightarrow \text{لا} \quad (\text{not } \text{لا})$$

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 FB50..FDFB and FE70..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.

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 `DerivedJoiningType.txt` and the `Joining_Group` values are extracted and listed in `DerivedJoiningGroup.txt`.

Dual-Joining. *Table 8-8* exemplifies dual-joining Arabic characters and illustrates the forms taken by the letter skeletons and their diacritical 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. These joining groups include `NYA`, `FARSI YEH`, and `BURUSHASKI YEH BARREE`.

Table 8-8. Dual-Joining Arabic Characters

Joining Group	X _n	X _r	X _m	X _l	Notes
BEH	ب	ب	ب	ب	Includes TEH and THEH.
NOON	ن	ن	ن	ن	
NYA	ث	ث	ث	ث	Jawi NYA.
YEH	ي	ي	ي	ي	Includes ALEF MAKSURA.
FARSI YEH	ی	ی	ی	ی	
BURUSHASKI YEH BARREE	ے	ے	ے	ے	Left-connecting form of YEH BARREE
HAH	ح	ح	ح	ح	Includes KHAH and JEEM.
SEEN	س	س	س	س	Includes SHEEN.
SAD	ص	ص	ص	ص	Includes DAD.
TAH	ط	ط	ط	ط	Includes ZAH.
AIN	ع	ع	ع	ع	Includes GHAIN.
FEH	ف	ف	ف	ف	

Table 8-8. Dual-Joining Arabic Characters (Continued)

Joining Group	X _n	X _r	X _m	X _l	Notes
QAF	ق	ق	قا	قا	
MEEM	م	م	ما	ما	
HEH	ه	ه	ها	ها	
KNOTTED HEH	ه	ه	ها	ها	
HEH GOAL	ه	ه	هـ	هـ	Includes HAMZA ON HEH GOAL.
KAF	ك	ك	كا	كا	
SWASH KAF	ك	ك	كا	كا	
GAF	گ	گ	گا	گا	
LAM	ل	ل	لا	لا	

Right-Joining. Table 8-9 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 8-9. Right-Joining Arabic Characters

Joining Group	X _n	X _r	Notes
ALEF	ا	ا	
WAW	و	و	
DAL	د	د	Includes THAL.
REH	ر	ر	Includes ZAIN.
TEH MARBUTA	ة	ة	Includes HAMZA ON HEH.
TEH MARBUTA GOAL	ة	ة	
YEH WITH TAIL	ي	ي	
YEH BARREE	ي	ي	
ROHINGYA YEH		ي	Isolated form does not occur.

In some cases, characters occur only at the end of words in correct spelling; they 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.

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 isolate forms of U+0647 ARABIC LETTER HEH and U+06C1 ARABIC LETTER HEH GOAL both look like U+06D5 ARABIC LETTER AE.

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 Koranic Arabic. It behaves differently from most Arabic letters, in a way surprising to 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 8-10.

Table 8-10. Forms of the Arabic Letter *yeh*

Character	Joining Group	X _n	X _r	X _m	X _l
U+0649 ALEF MAKSURA	YEH	ى	ي	ا	ا
U+064A YEH	YEH	ي	ي	ي	ي
U+06CC FARSI YEH	FARSI YEH	ى	ى	ي	ي
U+0777 YEH WITH DIGIT FOUR BELOW	YEH	ى	ي	ا	ا
U+0620 KASHMIRI YEH	YEH	ي	ي	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		ڤ		

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 8-10*. 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 8-10*. 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 8-10*.

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. It has the YEH joining group, with the shapes as shown in the fifth row of *Table 8-10*. However, when Kashmiri is written in Nastaliq style, the final and isolated forms of *kashmiri yeh* usually appear as truncated *yeh* shapes (*ﻯ*) without the diacritic ring.

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, but never occurs in isolated form. It only occurs after certain consonants, forming a conjunct letter with those consonants.

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 diacritic 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 Cameroun, to represent a palatal implosive.

In most other cases when a *hamza* is needed as a mark above for an extended Arabic letter, U+0654 ARABIC HAMZA ABOVE can be freely used in combination with basic Arabic letters.

Two exceptions are the extended Arabic letters U+0681 ARABIC LETTER HAH WITH HAMZA ABOVE and U+076C ARABIC LETTER REH WITH HAMZA ABOVE, where the *hamza* mark is functioning as an *ijam* (diacritic), rather than as a normal *hamza*. In those two cases, the extended Arabic letters have no canonical decompositions; consequently, the preference is to use those two precomposed forms, rather than applying U+0654 ARABIC HAMZA ABOVE to *hah* or to *reh*, respectively.

These interactions between various letters and the *hamza* are summarized in *Table 8-11*.

Table 8-11. 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
08A8	yeh with 2 dots below and hamza above	None

The first five entries in *Table 8-11* show the cases where the *hamza above* can be freely used, and where there is a canonical equivalence to the precomposed characters. The last three entries show the exceptions, where use of the *hamza above* is inappropriate, and where only the precomposed characters should be used.

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 [u:]. 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 *flap r*) and U+06B5 ARABIC LETTER LAM WITH SMALL V (for the Kurdish *velarized l*).

The Unicode Standard 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.

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 twentieth 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 ALEF and related characters.

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 Koranic annotation signs.

Arabic Presentation Forms-A: U+FB50–U+FDFF

This block contains a list of presentation forms (glyphs) encoded as characters for compatibility. As with most other compatibility encodings, these characters have a preferred encoding that makes use of noncompatibility 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. 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. These 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 *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+FB50..U+FB5F 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. Any use in juxtaposition with an Arabic letter skeleton is undefined.

The Arabic nukta symbols do not partake of any Arabic shaping behavior. 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.

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 the obligatory LAM-ALEF ligature. They are included here for compatibility with preexisting standards and legacy implementations that use these forms as characters. They can be replaced by letters from the Arabic block (U+0600..U+06FF). Implementations can 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 *Section 16.8, Specials*.

8.3 Syriac

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.*

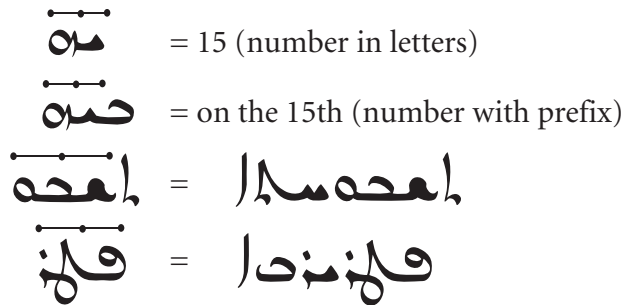
Character Names. Character names follow the East Syriac convention for naming the letters of the alphabet. Diacritical points use a descriptive naming—for example, SYRIAC DOT ABOVE.

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 diacritic 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.

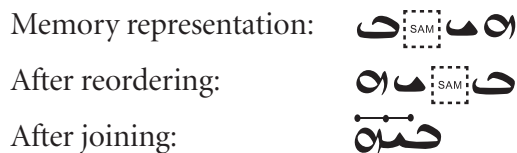
The Syriac abbreviation is used for letter numbers and contractions. A Syriac abbreviation generally extends from the last tall character in the word until the end of the word. A common exception to this rule is found with letter numbers that are preceded by a preposition character, as seen in *Figure 8-7*.

Figure 8-7. Syriac Abbreviation



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 8-8*.

Figure 8-8. Use of SAM



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.

Diacritic Marks and Vowels. The function of the diacritic 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.

Diacritic 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 8-12*.

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 *hara-kat* are used in Garshuni to indicate the corresponding Arabic vowels and diacritics.

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 char-

Table 8-12. 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	These points are used for various purposes—grammatical, phonological, and otherwise. They differ typographically and semantically from the <i>qushshaya</i> , <i>rukkakha</i> points, and the dotted vowel points.
U+0308	This is the plural marker. It is also used in Garshuni for the Arabic <i>teh marbuta</i> .
U+030A, U+0325	These are two other forms for the indication of <i>qushshaya</i> and <i>rukkakha</i> . They are used interchangeably with U+0741 SYRIAC QUSHSHAYA and U+0742 SYRIAC RUKKAKHA, especially in West Syriac grammar books.
U+0324	This diacritic mark is found in ancient manuscripts. It has a grammatical and phonological function.
U+032D	This is one of the <i>digit markers</i> .
U+032E	This is a mark used in late and modern East Syriac texts as well as in Swadaya to indicate a fricative <i>pe</i> .

acters. The basic joining types are identical to those specified for the Arabic script. 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 Table 8-13.

Table 8-13. Syriac Final Alaph Glyph Types

Glyph Type	Description
A _{fi}	Final joining (alaph only)
A _{fn}	Final non-joining <i>except</i> 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 Section 8.2, *Arabic*. 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 to its left will take the form of A_{fi}.

$$\{ + \text{Ⲁ} \rightarrow \} + \text{Ⲁ} \rightarrow \text{Ⲁ}$$

- 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 to its left will take the form of A_{fn}.

$$\{ + \text{Ⲁ} \rightarrow \} + \text{Ⲁ} \rightarrow \text{Ⲁ}$$

S3 An alaph that has a character with *Joining_Group*=*Dalath_Rish* to its right and a non-joining character to its left will take the form of *A_{fx}*.

$$\text{ܐ} + \text{ܕ} \rightarrow \text{ܐ} + \text{ܕ} \rightarrow \text{ܕܐ}$$

The example in rule S3 is shown in the East Syriac font style.

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 8.2, Arabic*.

Table 8-14 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 8-14. Dual-Joining Syriac Characters

Joining Group	X _n	X _r	X _m	X _l	Notes
Beth	ܐ	ܐ	ܐ	ܐ	Includes PERSIAN BHETH
Gamal	ܘ	ܘ	ܘ	ܘ	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	ܚ	ܚ	ܚ	ܚ	
Khaph	ܚ	ܚ	ܚ	ܚ	Sogdian
Lamadh	ܠ	ܠ	ܠ	ܠ	
Mim	ܡ	ܡ	ܡ	ܡ	
Nun	ܢ	ܢ	ܢ	ܢ	
Semkath	ܦ	ܦ	ܦ	ܦ	
Final_Semkath	ܦ	ܦ	ܦ	ܦ	
E	ܐ	ܐ	ܐ	ܐ	
Pe	ܦ	ܦ	ܦ	ܦ	
Reversed_Pe	ܦ	ܦ	ܦ	ܦ	
Fe	ܦ	ܦ	ܦ	ܦ	Sogdian
Qaph	ܩ	ܩ	ܩ	ܩ	
Shin	ܫ	ܫ	ܫ	ܫ	

Table 8-15 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 8-15. Right-Joining Syriac Characters

Joining Group	X _n	X _r	Notes
Dalath_Rish	ܕ	ܕ	Includes RISH, DOTLESS DALATH RISH, and PERSIAN DHALATH
He	ܗ	ܗ	
Syriac_Waw	ܘ	ܘ	
Zain	ܙ	ܙ	
Zhain	ܙ	ܙ	Sogdian
Yudh_He	ܘܗ	ܘܗ	
Sadhe	ܫ	ܫ	
Taw	ܬ	ܬ	

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 8-16 illustrates all of the glyph forms for *alaph* in each of the three major Syriac type styles.

Table 8-16. 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 8-17 identifies the principal valid ligatures for each font style. When applicable, these ligatures are obligatory, unless denoted with an asterisk (*).

Table 8-17. 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

Table 8-17. Syriac Ligatures (Continued)

Characters	Estrangela	Serto (West Syriac)	East Syriac	Sources
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*	

8.4 Samaritan

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 14.10, Phoenician*). In contrast, the more recent square Hebrew script associated with Judaism derives from the Imperial Aramaic script (see *Section 14.11, 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-ALEF, 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 8-18*.

Table 8-18. Samaritan Performative Punctuation Marks

Code Point	Name	Description
0833	<i>bau</i>	request, prayer, humble petition
0834	<i>atmaau</i>	expression of surprise
0835	<i>shiyyaalaa</i>	question
0838	<i>ziqaa</i>	shout, cry
083A	<i>zaef</i>	outburst indicating vehemence or anger
083B	<i>turu</i>	didactic expression, a “teaching”
083C	<i>arkaanu</i>	expression of submissiveness

8.5 Thaana

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 twentieth century. Loan words

from Arabic may be written in the Arabic script, although this custom is not very prevalent today. (See *Section 8.2, 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 8-19*.

Table 8-19. Thaana Glyph Placement

Syllable	Display
<i>tha</i>	ٲ
<i>thaa</i>	ٲٲ
<i>thi</i>	ٲٲ
<i>thee</i>	ٲٲٲ
<i>thu</i>	ٲٲٲ
<i>thoo</i>	ٲٲٲ
<i>the</i>	ٲٲ
<i>they</i>	ٲٲٲ
<i>tho</i>	ٲٲ
<i>thoa</i>	ٲٲ
<i>th</i>	ٲ

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*.

Chapter 9

South Asian Scripts-I

The following South Asian scripts are described in this chapter:

<i>Devanagari</i>	<i>Gujarati</i>	<i>Telugu</i>
<i>Bengali</i>	<i>Oriya</i>	<i>Kannada</i>
<i>Gurmukhi</i>	<i>Tamil</i>	<i>Malayalam</i>

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 *pulli* 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, Bengali, 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 all nine 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.

9.1 Devanagari

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–F4₁₆ in the ISCII-1988 standard. The same character code layout is followed for eight other Indic scripts in the Unicode Standard: Bengali, 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)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.

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 represents 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 to depict the initial consonant in consonant clusters. 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.)

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, 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 9-1* 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.

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.

Table 9-1. Devanagari Vowel Letters

For	Use	Do Not Use
ऐ	0904	<0905, 0946>
आ	0906	<0905, 093E>
ई	0908	<0930, 094D, 0907>
ऊ	090A	<0909, 0941>
ऍ	090D	<090F, 0945>
ऐ	090E	<090F, 0946>
ऐ	0910	<090F, 0947>
आँ	0911	<0905, 0949> or <0906, 0945>
ओ	0912	<0905, 094A> or <0906, 0946>
ओ	0913	<0905, 094B> or <0906, 0947>
औ	0914	<0905, 094C> or <0906, 0948>
अँ	0972	<0905, 0945>
अ	0973	<0905, 093A>
आँ	0974	<0905, 093B> or <0906, 093A>
औ	0975	<0905, 094F>
अु	0976	<0905, 0956>
अु	0977	<0905, 0957>

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 9-1.

Figure 9-1. Dead Consonants in Devanagari

$$TA_n + VIRAMA_n \rightarrow TA_d$$

$$त + ः \rightarrow त्$$

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 9-2*).

Figure 9-2. Conjunct Formations in Devanagari

- | | |
|---|---|
| (1) $GA_d + DHA_l \rightarrow GA_h + DHA_n$ | (3) $KA_d + SSA_l \rightarrow K.SSA_n$ |
| $ग् + ध \rightarrow र्ध$ | $क् + ष \rightarrow क्ष$ |
| (2) $KA_d + KA_l \rightarrow K.KA_n$ | (4) $RA_d + KA_l \rightarrow KA_l + RA_{sup}$ |
| $क् + क \rightarrow क्क$ | $र् + क \rightarrow र्क$ |

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 9-2*, the dead consonant RA_d is depicted with the nonspacing combining mark RA_{sup} (*repha*).

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.

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 9-3*, the use of ZERO WIDTH NON-JOINER prevents the default formation of the conjunct form $क्ष$ (K.SSA_n).

Figure 9-3. Preventing Conjunct Forms in Devanagari

$$KA_d + ZWNJ + SSA_l \rightarrow KA_d + SSA_n$$

$$क् + \text{ZWNJ} + ष \rightarrow क् ष$$

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 9-4*.

Figure 9-4. Half-Consonants in Devanagari

$$KA_d + ZWJ + SSA_l \rightarrow KA_h + SSA_n$$

$$\text{क्} + \boxed{\text{ZWJ}} + \text{ष} \rightarrow \text{क्ष}$$

In the absence of the ZERO WIDTH JOINER, the sequence in *Figure 9-4* 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 letterform. That is, this technique may be used to encode independent half-forms, as shown in *Figure 9-5*.

Figure 9-5. Independent Half-Forms in Devanagari

$$GA_d + ZWJ \rightarrow GA_h$$

$$\text{ग्} + \boxed{\text{ZWJ}} \rightarrow \text{र्}$$

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 9-6*.

Figure 9-6. Half-Consonants in Oriya

$$KA_n + ZWJ + VIRAMA + TA_l \rightarrow KA_l + TA_h$$

$$\text{କ୍} + \boxed{\text{ZWJ}} + \text{ଂ} + \text{ଟ} \rightarrow \text{କ୍ଟ}$$

In the absence of the ZERO WIDTH JOINER, the sequence in *Figure 9-6* would normally produce the full conjunct form କ୍‌ଟ (K.TA_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 9-7*).

Figure 9-7. Consonant Forms in Devanagari and Oriya

क + ष	→ कष	KA _l + SSA _n
क + ◌ + ष	→ क्ष	K.SSA _n
क + ◌ + ZW + ष	→ क्ष	KA _h + SSA _n
क + ◌ + ZW NJ + ष	→ क्ष	KA _d + SSA _n
ଜ + ଷ + ଢ	→ ଜ୍ଠ	K.TA _n
ଜ + ZW + ଷ + ଢ	→ ଜ୍ଠ	KA _n + TA _h
ଜ + ◌ + ZW NJ + ଢ	→ ଜ୍ଠ	KA _d + TA _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.

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 <i>repha</i> .
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 .

$$TA_n + VIRAMA_n \rightarrow TA_d$$

$$त + ः \rightarrow त्$$

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.

$$RA_d + KA_l \rightarrow KA_l + RA_{sup} \quad \text{Displayed Output}$$

$$\underset{\sim}{र} + क \rightarrow क + ः \rightarrow क्$$

$$RA_d^1 + RA_d^2 \rightarrow RA_d^2 + RA_{sup}^1$$

$$\underset{\sim}{र} + \underset{\sim}{र} \rightarrow \underset{\sim}{र} + ः \rightarrow \underset{\sim}{र}$$

- 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.

$$RA_d + JA_d + NYA_l \rightarrow J.NYA_n + RA_{sup} \quad \text{Displayed Output}$$

$$\underset{\sim}{र} + ज् + ज \rightarrow ज्ञ + ः \rightarrow ज्ञ्$$

- 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.

$$RA_d + GA_d + GHA_l \rightarrow GA_h + GHA_l + RA_{sup} \quad \text{Displayed Output}$$

$$\text{र् + ग् + घ} \rightarrow \text{ग + घ + ँ} \rightarrow \text{गघ}$$

- 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.

$$RRA_n + VIRAMA_n \rightarrow RRA_h$$

$$\text{र् + ्} \rightarrow \text{ः}$$

- 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} .

$$RA_d + ZWJ \rightarrow RA_h$$

$$\text{र् + } \boxed{\text{ZWJ}} \rightarrow \text{ः}$$

- 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 .

$$TTHA_d + RA_l \rightarrow TTHA_n + RA_{sub} \quad \text{Displayed Output}$$

$$\text{ठ् + र} \rightarrow \text{ठ + ्र} \rightarrow \text{ठ्र}$$

- 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.

$$PHA_d + RA_l \rightarrow PHA_n + RA_{sub} \quad \text{Displayed Output}$$

$$\text{फ् + र} \rightarrow \text{फ + ्र} \rightarrow \text{फ्र}$$

- 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.*

$$TA_d + RA_d \rightarrow TA_n + RA_{sub} + VIRAMA_n \rightarrow T.RA_d$$

$$\underline{त्} + \underline{र्} \rightarrow \underline{त} + \overset{\circ}{\underline{र}} + \overset{\circ}{\underline{्}} \rightarrow \underline{त्र}$$

A dead consonant conjunct form that contains an absorbed RA_d may subsequently combine to form a multipart conjunct form.

$$T.RA_d + YA_l \rightarrow T.R.YA_n$$

$$\underline{त्र} + \underline{य} \rightarrow \underline{त्रय}$$

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*.

- 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.*

$$KA_n + NUKTA_n + VIRAMA_n \rightarrow QA_d$$

$$\underline{क} + \overset{\circ}{\underline{्}} + \overset{\circ}{\underline{्}} \rightarrow \underline{क्}$$

- 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. The relative placement of these marks is horizontal rather than vertical; the horizontal rendering order may vary according to typographic concerns.*

$$KA_n + AA_{vs} + CANDRABINDU_n$$

$$\underline{क} + \overset{\circ}{\underline{ा}} + \overset{\circ}{\underline{ँ}} \rightarrow \underline{काँ}$$

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.*

$$JA_d + NYA_l \rightarrow J.NYA_n$$

$$TTA_d + TTHA_l \rightarrow TT.TTHA_n$$

$$\underline{ज्} + \underline{ञ} \rightarrow \underline{ज्ञ}$$

$$\underline{ट्} + \underline{ठ} \rightarrow \underline{ट्ठ}$$

R12 A conjunct ligature form can itself behave as a dead consonant and enter into further, more complex ligatures.

$$SA_d + TA_d + RA_n \rightarrow SA_d + T.RA_n \rightarrow S.T.RA_n$$

$$\text{स्} + \text{त्} + \text{र} \rightarrow \text{स्} + \text{त्र} \rightarrow \text{स्त्र}$$

A conjunct ligature form can also produce a half-form.

$$K.SSA_d + YA_l \rightarrow K.SS_h + YA_n$$

$$\text{क्ष} + \text{य} \rightarrow \text{क्ष्य}$$

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).

$$KA_n + RA_{sub} \rightarrow K.RA_n$$

$$PHA_n + RA_{sub} \rightarrow PH.RA_n$$

$$\text{क} + \text{्} \rightarrow \text{क्र}$$

$$\text{फ} + \text{्} \rightarrow \text{फ्र}$$

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} .

$$RA_l + U_{vs} \rightarrow RU_n$$

$$RA_l + UU_{vs} \rightarrow RUU_n$$

$$\text{र} + \text{ु} \rightarrow \text{रु}$$

$$\text{र} + \text{ू} \rightarrow \text{रू}$$

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 9-8).

Figure 9-8. Rendering Order in Devanagari

Character Order

Glyph Order

$$KA_n + I_{vs} \rightarrow I + KA_n$$

$$\text{क} + \text{ि} \rightarrow \text{कि}$$

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 l_{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.

$$TA_d + RA_l + l_{vs} \rightarrow T.RA_n + l_{vs} \rightarrow l_{vs} + T.RA_d$$

$$त् + र + ि \rightarrow त्र + ि \rightarrow त्रि$$

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 l_{vs} is rendered after the rightmost such explicit virama.

$$TA_d + ZWNJ + RA_l + l_{vs} \rightarrow TA_d + l_{vs} + RA_l$$

$$त् + \boxed{\text{ZWJ}} + र + ि \rightarrow त्रि$$

Sample Half-Forms. Table 9-2 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 9-2. Sample Devanagari Half-Forms

क + ् + $\boxed{\text{ZWJ}}$ → क्	न + ् + $\boxed{\text{ZWJ}}$ → न्
ख + ् + $\boxed{\text{ZWJ}}$ → ख्	प + ् + $\boxed{\text{ZWJ}}$ → प्
ग + ् + $\boxed{\text{ZWJ}}$ → ग्	फ + ् + $\boxed{\text{ZWJ}}$ → फ्
घ + ् + $\boxed{\text{ZWJ}}$ → घ्	ब + ् + $\boxed{\text{ZWJ}}$ → ब्
च + ् + $\boxed{\text{ZWJ}}$ → च्	भ + ् + $\boxed{\text{ZWJ}}$ → भ्
ज + ् + $\boxed{\text{ZWJ}}$ → ज्	म + ् + $\boxed{\text{ZWJ}}$ → म्
झ + ् + $\boxed{\text{ZWJ}}$ → झ्	य + ् + $\boxed{\text{ZWJ}}$ → य्
ञ + ् + $\boxed{\text{ZWJ}}$ → ञ्	ल + ् + $\boxed{\text{ZWJ}}$ → ल्
ण + ् + $\boxed{\text{ZWJ}}$ → ण्	व + ् + $\boxed{\text{ZWJ}}$ → व्
त + ् + $\boxed{\text{ZWJ}}$ → त्	श + ् + $\boxed{\text{ZWJ}}$ → श्
थ + ् + $\boxed{\text{ZWJ}}$ → थ्	ष + ् + $\boxed{\text{ZWJ}}$ → ष्
ध + ् + $\boxed{\text{ZWJ}}$ → ध्	स + ् + $\boxed{\text{ZWJ}}$ → स्

Sample Ligatures. Table 9-3 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 9-3. 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
ड + ् + क → ड्क	ड + ् + ढ → ड्ढ
ड + ् + ख → ड्ख	त + ् + त → त्त
ड + ् + ग → ड्ग	त + ् + र → त्र
ड + ् + घ → ड्घ	न + ् + न → न्न
ज + ् + ज → ज्ज	फ + ् + र → फ्र
ज + ् + ज → ज्ज	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 + ् → ह
द + ् + म → द्म	र + ् → र
द + ् + य → द्य	र + ् → रू
द + ् + व → द्व	र + ् → र्र
ट + ् + ट → ट्ट	स + ् + त्र → स्त्र

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 9-4. 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.

Language-Specific Allographs. In Marathi and some South Indian orthographies, variant glyphs are preferred for U+0932 DEVANAGARI LETTER LA and U+0936 DEVANAGARI LETTER SHA, as shown in Figure 9-9. Marathi also makes use of the “eyelash” form of the letter RA, as discussed in rule R5.

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 repre-

Table 9-4. 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

Figure 9-9. Marathi Allographs

	Normal	Marathi		Normal	Marathi
LA	ल	ल	SHA	श	श
	U+0932			U+0936	

sented 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..U+0954 are a set of combining marks used in transcription of Sanskrit texts.

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 Bengali, Gujarati, and so on. 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 10.2, Tibetan* for further information.

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 *anudatta*, 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.

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 9-10*. The Maithili examples show the same sentence, first in full form, and then using U+02BC to show truncation of words.

Figure 9-10. 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 9-11*.

Figure 9-11. Use of Avagraha in Dogri

Example	Meaning
तला तलाऽ	sole pond

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 9-1*. A few of the letters identified as being used for Kashmiri are also used to write the Bihari languages.

Letters for Bihari Languages. A number of the Devanagari vowel letters have been used to write the Bihari languages Bhojputi, Magadhi, and Maithili, as listed in *Table 9-5*.

Table 9-5. Devanagari Vowels Used in Bihari Languages

U+090E	DEVANAGARI LETTER SHORT E
U+0912	DEVANAGARI LETTER SHORT O
U+0946	DEVANAGARI VOWEL SIGN SHORT E
U+094A	DEVANAGARI VOWEL SIGN SHORT O
U+0973	DEVANAGARI LETTER OE
U+0974	DEVANAGARI LETTER OOE
U+0975	DEVANAGARI LETTER AW
U+093A	DEVANAGARI VOWEL SIGN OE
U+093B	DEVANAGARI VOWEL SIGN OOE
U+094F	DEVANAGARI VOWEL SIGN AW

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 9-6* 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 9-6. Prishthamatra Orthography

	Prishthamatra Orthography	Modern Orthography
ke	क <0915, 094E>	के <0915, 0947>
kai	के <0915, 094E, 0947>	कै <0915, 0948>
ko	का <0915, 094E, 093E>	को <0915, 094B>
kau	को <0915, 094E, 094B>	कौ <0915, 094C>

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 marks of nasalization (*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 *ekāśruti* (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 Rāṇyāniya 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-Samhitā*) 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 Rāṇyāniya 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 DEVANAGARI 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 Jaiminiya 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.

Marks of Nasalization. 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 syllables 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.

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*. Nasal

characters are also included in the block. U+1CD3 VEDIC SIGN NIHSHVASA indicates where a breath may be taken. Finally, the block includes U+1CF2 VEDIC SIGN ARDHAVISARGA.

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 *jihvāmūliya*, a velar fricative occurring only before the unvoiced velar stops *ka* and *kha*, or the *upadhmāniya*, 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.

9.2 Bengali (Bangla)

Bengali: U+0980–U+09FF

Scripts encoded in the Unicode Standard often are used to write many languages. The script termed *Bengali* in Unicode is no exception. It is used for writing languages such as Bengali, Assamese, Bishnupriya Manipuri, Daphla, Garo, Hallam, Khasi, Mizo, Munda, Naga, Rian, and Santali. In the Indian state of West Bengal and the People's Republic of

Bangladesh, the preferred name for the Bengali script and language is *Bangla*. In the Indian state of Assam, the preferred name for the script is *Asamiya* or *Assamese*. 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 Bengali, with the addition of extra characters. The Unicode Bengali block fully supports modern Assamese orthography.

The Bengali script is a North Indian script closely related to Devanagari.

Virama (Hasant). The Bengali script uses the Unicode virama model to form conjunct consonants. In Bengali, the virama is known as *hasant*.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 9-7* 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 9-7. Bengali Vowel Letters

For	Use	Do Not Use
আ	0986	<0985, 09BE>
ঐ	09E0	<098B, 09C3>
ঔ	09E1	<098C, 09E2>

Two-Part Vowel Signs. The Bengali script, along with a number of other Indic scripts, makes use of two-part vowel signs. In these vowels 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 Bengali 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 Bengali 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 Bengali script.

Characters for Assamese. U+09F0 BENGALI LETTER RA WITH MIDDLE DIAGONAL and U+09F1 BENGALI LETTER RA WITH LOWER DIAGONAL are characters in this block required to write Assamese.

Rendering Behavior. Like other Brahmic scripts in the Unicode Standard, Bengali uses the *hasant* to form conjunct characters. For example, U+09B8 ষ BENGALI LETTER SA + U+09CD ঙ BENGALI SIGN VIRAMA + U+0995 ঞ BENGALI LETTER KA yields the conjunct ঞ SKA. For general principles regarding the rendering of the Bengali script, see the rules for rendering in *Section 9.1, Devanagari*.

Consonant-Vowel Ligatures. Some Bengali 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 9-8*.

Table 9-8. Bengali Consonant-Vowel Combinations

	Code Points	Ligated	Non-ligated
<i>gu</i>	<0997, 09C1>	গু	গূ
<i>ru</i>	<09B0, 09C1>	রু	রূ
<i>rū</i>	<09B0, 09C2>	রু	রূ
<i>śu</i>	<09B6, 09C1>	শু	শূ
<i>hu</i>	<09B9, 09C1>	হু	হূ
<i>hr</i>	<09B9, 09C3>	হ্র	হ্র

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 Bengali 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 16.2, 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 9-12*.

Figure 9-12. Requesting Bengali Consonant-Vowel Ligature

গ + ◌ → গূ
0997 09C1

গ + ◌ + ◌ → গু
0997 200D 09C1

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 9-13*.

Figure 9-13. Blocking Bengali Consonant-Vowel Ligature

গ + ◌ → গু
0997 09C1

গ + ◌ + ◌ → গূ
0997 200C 09C1

Khiya. The letter ঞ, known as *khiya*, is often considered as a distinct letter of the Bengla alphabet. However, it is not encoded separately. It is represented by the sequence <U+0995 ঞ BENGALI LETTER KA, U+09CD ◌ BENGALI SIGN VIRAMA, U+09B7 ষ BENGALI LETTER SSA>.

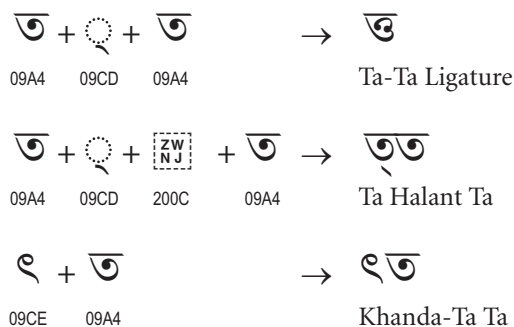
Khanda Ta. In Bengali, a dead consonant *ta* makes use of a special form, U+09CE BENGALI LETTER KHANDA TA. This form is used in all contexts except where it is immediately followed by one of the consonants: *ta*, *tha*, *na*, *ba*, *ma*, *ya*, or *ra*.


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 BENGALI LETTER TA, U+09CD BENGALI SIGN VIRAMA, U+200D ZERO WIDTH JOINER> in all circumstances. U+09CE BENGALI LETTER KHANDA TA should instead be used explicitly in newly generated text, but users are cautioned that instances of the older representation may exist.

The Bengali 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 9-14*.






Figure 9-14. Bengali 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 . 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 ZERO WIDTH NON-JOINER is inserted after the *hasant*. That sequence is always displayed with the explicit *hasant*, as shown in the second example in *Figure 9-14*.


When the syllable *tta* is written with a *khanda ta*, however, the character U+09CE BENGALI LETTER KHANDA TA 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 9-14*.

Ya-phalaa. *Ya-phalaa* is a presentation form of U+09AF  BENGALI LETTER YA. Represented by the sequence <U+09CD  BENGALI SIGN VIRAMA, U+09AF  BENGALI LETTER YA>, *ya-phalaa* has a special form . When combined with U+09BE  BENGALI VOWEL SIGN AA, it is used for transcribing [æ] as in the “a” in the English word “bat.” *Ya-phalaa* can be applied to initial vowels as well:

 = <0985, 09CD, 09AF, 09BE> (*a- hasant ya -aa*)

 = <098E, 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:

 = <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 Section 9.1, *Devanagari*, “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*>.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with Bengali are found in the Devanagari block; see Section 9.1, *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 9-9.

Table 9-9. Use of Apostrophe in Bangla

Example	Meaning
ক'রে	after, on doing (something)
প'রে	} above
ওপরে	

9.3 Gurmukhi

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 GURMUKHI LETTER SHA and U+0A33 GURMUKHI LETTER LLA, but do not include U+0A5C 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 ੲ 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. 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.

Punjabi does not have complex combinations of consonant sounds. Furthermore, the orthography is not strictly phonetic, and sometimes the inherent /a/ sound is not pronounced. For example, the word ਗੁਰਮੁਖੀ *gurmukhī* is represented with the sequence <0A17, 0A41, 0A30, 0A2E, 0A41, 0A16, 0A40>, which could be transliterated as *gur-amukhī*; this lack of pronunciation is systematic at the end of a word. As a result, the virama sign is seldom used with the Gurmukhi script.

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 ੁ attached to ੳ, as in ਉਮਾਰਾ and both the vowel signs ੱ and ੲ attached to a consonant, as in ਗੋਬਿੰਦ *goubida*; 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 ਗਿਆਨ *ghiana*.

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.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. Table 9-10 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 9-10. 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 GURMUKHI URA and U+0A72 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 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 9.1, *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 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 GURMUKHI LETTER HA and U+0A35 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 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 9-11*.

Table 9-11. Gurmukhi Conjuncts

ਮ	+	੍	+	ਹ	→	ਮ੍ਹ	(<i>mha</i>)	pairin ha
ਪ	+	੍	+	ਰ	→	ਪ੍ਰ	(<i>pra</i>)	pairin ra
ਦ	+	੍	+	ਵ	→	ਦ੍ਵ	(<i>dva</i>)	pairin va
ਦ	+	੍	+	ਯ	→	ਦਯ	(<i>dya</i>)	addha ya

Other letters behaved similarly in old inscriptions, as shown in *Table 9-12*.

Table 9-12. Additional Pairin and Addha Forms in Gurmukhi

ਸ	+	੍	+	ਗ	→	ਸ੍ਗ	(<i>sga</i>)	pairin ga
ਸ	+	੍	+	ਚ	→	ਸ੍ਚ	(<i>sca</i>)	pairin ca
ਸ	+	੍	+	ਟ	→	ਸ੍ਟ	(<i>stta</i>)	pairin tta
ਸ	+	੍	+	ਠ	→	ਸ੍ਠ	(<i>sttha</i>)	pairin ttha
ਸ	+	੍	+	ਤ	→	ਸ੍ਤ	(<i>sta</i>)	pairin ta
ਸ	+	੍	+	ਦ	→	ਸ੍ਦ	(<i>sda</i>)	pairin da
ਸ	+	੍	+	ਨ	→	ਸ੍ਨ	(<i>sna</i>)	pairin na
ਸ	+	੍	+	ਥ	→	ਸ੍ਥ	(<i>stha</i>)	pairin tha
ਸ	+	੍	+	ਯ	→	ਸ੍ਯ	(<i>sya</i>)	pairin ya
ਸ	+	੍	+	ਥ	→	ਸਥ	(<i>stha</i>)	addha tha
ਸ	+	੍	+	ਮ	→	ਸਮ	(<i>sma</i>)	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 9-13*. The reduced form of an initial U+0A30 GURMUKHI LETTER RA is similar to the Devanagari superscript RA (*repha*), but this usage is rare, even in older texts.

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 GURMUKHI LETTER NA and U+0A20 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*.

Table 9-13. Use of Joiners in Gurmukhi

ਸ	+	੍	+	ਵ	→	ਸ੍ਵ	(<i>sva</i>)
ਰ	+	੍	+	ਵ	→	ਰ੍ਵ	(<i>rva</i>)
ਸ	+	੍	+	␣	+	ਵ	→ ਸ੍ਵ (sva)
ਰ	+	੍	+	␣	+	ਵ	→ ਰ੍ਵ (rva)
ਸ	+	੍	+	␣	+	ਵ	→ ਸ੍ਵ (sva)
ਰ	+	੍	+	␣	+	ਵ	→ ਰ੍ਵ (rva)

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 GURMUKHI EK ONKAR, which is also a religious symbol, can have different presentation forms, which do not change its meaning. The font used in the code charts shows a highly stylized form; simpler forms look like the digit one, followed by a sign based on *ura*, along with a long upper tail.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with Gurmukhi are found in the Devanagari block. See *Section 9.1, Devanagari*, for more information. Punjabi also uses Latin punctuation.

9.4 Gujarati

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 letterforms, 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 9-14* 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 9-14. Gujarati Vowel Letters

For	Use	Do Not Use
અ	0A86	<0A85, 0ABE>
એ	0A8D	<0A85, 0AC5>
એ	0A8F	<0A85, 0AC7>
એ	0A90	<0A85, 0AC8>
ઓ	0A91	<0A85, 0AC9>
ઓ	0A93	<0A85, 0ACB> or <0A85, 0ABE, 0AC5>
ઓ	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 9.1, *Devanagari*. Like other Brahmic scripts in the Unicode Standard, Gujarati uses the virama to form conjunct characters. The virama is informally called *khodo*, 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 9-15.

Table 9-15. Gujarati Conjuncts

ક	+	◌̣	+	૫	→	ક૫	(kṣa)
જ	+	◌̣	+	ઞ	→	જઞ	(jñā)
ટ	+	◌̣	+	૫	→	ટ૫	(tṣa)
ટ	+	◌̣	+	ટ	→	ટટ	(ṭṭa)
ર	+	◌̣	+	ક	→	રક	(rka)
ક	+	◌̣	+	ર	→	કર	(kra)

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 9.1, *Devanagari*.

9.5 Oriya

Oriya: U+0B00–U+0B7F

The Oriya script is a North Indian script that is structurally similar to Devanagari, but with semicircular lines at the top of most letters instead of the straight horizontal bars of Devanagari. The actual shapes of the letters, particularly for vowel signs, show similarities to Tamil. The Oriya script is used to write the Oriya language of the Orissa state in India as well as minority languages such as Khondi and Santali.

Special Characters. U+0B57 ORIYA AU LENGTH MARK is provided as an encoding for the right side of the surroundrant 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 9-16 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 9-16. Oriya Vowel Letters

For	Use	Do Not Use
ଌ	0B06	<0B05, 0B3E>
ୠ	0B10	<0B0F, 0B57>
ୡ	0B14	<0B13, 0B57>

Rendering Behavior. For rendering of the Oriya script, see the rules for rendering in Section 9.1, *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 9-17*.

Table 9-17. Oriya Conjuncts

କ	+	୍	+	କ୍ଷ	→	କ୍ଷ	(<i>kṣa</i>)
କ	+	୍	+	ତ	→	କ୍ତ	(<i>кта</i>)
ତ	+	୍	+	କ	→	ତ୍କ	(<i>tka</i>)
ତ	+	୍	+	ୟ	→	ତ୍ୟ	(<i>tya</i>)

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 + ୠ + 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. See *Table 9-18*.

Table 9-18. Oriya Vowel Placement

କ	+	ା	→	କା	(<i>kā</i>)
କ	+	ି	→	କି	(<i>ki</i>)
କ	+	ି	→	କି	(<i>kī</i>)
କ	+	ୁ	→	କୁ	(<i>ku</i>)
କ	+	ୁ	→	କୁ	(<i>kū</i>)
କ	+	ୃ	→	କୃ	(<i>kṛ</i>)
କ	+	େ	→	କେ	(<i>ke</i>)
କ	+	ୈ	→	କୈ	(<i>kai</i>)
କ	+	ୋ	→	କୋ	(<i>ko</i>)
କ	+	ୌ	→	କୌ	(<i>kau</i>)

U+0B01 ORIYA SIGN CANDRABINDU is used for nasal vowels:

କ + ୠ → କାଁ (*kaṁ*)

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 ଚ 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 ଢ is sometimes found for *va*; in others, ଘ and ଞ have been shown, which in a more modern type style would be ଘ. 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 ୠ virama + U+0B2C ବ *ba* → କ୍ୱ [kwa]

U+0B2E ମ *ma* + U+0B4D ୠ virama + U+0B2C ବ *ba* → ମ୍ବ [mba]

The extended Oriya letter U+0B71 ଝ 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 ୞, ୟ, and ୟ.

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 9.1, Devanagari*. The mark U+0B70 ORIYA ISSHAR is placed before names of persons who are deceased.

Fraction Characters. As for many other scripts of India, Oriya has characters used to denote fractional 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.

9.6 Tamil

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 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 Tamil script also avoids the use conjunct consonant forms, although a few conventional conjuncts are used.

Virama (Pulḷi). 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, ZERO WIDTH NON-JOINER is used to display a visible virama, as shown previously in the Devangari example in *Figure 9-3*.

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 TAMIL SIGN VIRAMA, C2, ...>) is normally displayed with explicit viramas, which are called *pulli* in Tamil. The *pulli* 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 TAMIL SIGN ANUSVARA.

The conjuncts *kssa* and *shrii* are traditionally displayed by conjunct ligatures, as illustrated for *kssa* in *Figure 9-15*, but nowadays tend to be displayed using an explicit *pulli* as well.

Figure 9-15. Kssa Ligature in Tamil

க + ற் + க் → க்ஷ kṣa

To explicitly display a *pulli* for such sequences, ZERO WIDTH NON-JOINER can be inserted after the *pulli* 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.

Tamil Vowels

Independent Versus Dependent Vowels. In the Tamil script, the dependent vowel signs are not equivalent to a sequence of *virama* + *independent vowel*. For example:

ஊ + ி ≠ ஊ + ற் + 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 *Table 9-19*.

Table 9-19. Tamil Vowel Reordering

Memory Representation		Display
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 9-16*.

Figure 9-16. Tamil Two-Part Vowels

$$\begin{aligned} \text{ொ} \text{ா} \text{ } 0BCA &\equiv \text{ெ} \text{ } + \text{ா} \text{ } 0BC6 + 0BBE \\ \text{ோ} \text{ } 0BCB &\equiv \text{ே} \text{ } + \text{ா} \text{ } 0BC7 + 0BBE \\ \text{ெ} \text{ள} \text{ } 0BCC &\equiv \text{ெ} \text{ } + \text{ள} \text{ } 0BC6 + 0BD7 \end{aligned}$$

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. Note that the ூள in the third example is *not* U+0BB3 TAMIL LETTER LLA; it is U+0BD7 TAMIL AU LENGTH MARK.

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 Table 9-20.

Table 9-20. Tamil Vowel Splitting and Reordering

Memory Representation			Display
க	ொ		கொ
க	ெ	ா	கொ
க	ோ		கோ
க	ே	ா	கோ
க	ெள		கௌ
க	ெ	ள	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 9-17.

Figure 9-17. Vowel Reordering Around a Tamil Conjunct

$$\text{க} + \text{்} + \text{ஷ} + \text{ெ} + \text{ா} \rightarrow \text{கெஷா } k_{\text{šo}}$$

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.

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 *pulli*.

Ligatures with Vowel i. The vowel signs *i* ீ and *ii* ு form ligatures with the consonant *tta* ் as shown in examples 1 and 2 of Figure 9-18. 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 9-18.

Figure 9-18. Tamil Ligatures with *i*

- ① ட + ி → டி *ti*
- ② ட + ீ → டீ *tī*
- 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 9-21. 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 9-21. Tamil Ligatures with *u*

x	$x + \text{ி}$	$x + \text{஁}$
க	கு	கூ
ங	ஙு	கூ
ச	சு	சூ
ஞ	ஞு	ஞூ
ட	டு	டூ
ண	ணு	ணூ
த	து	தூ
ந	நு	நூ
ன	னு	னூ

x	$x + \text{ி}$	$x + \text{஁}$
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, வ், ஸ, ஹ, and the conjunct ச்ஷ, the vowel signs u ி and uu ு take a distinct spacing form, as shown in Figure 9-19.

Figure 9-19. Spacing Forms of Tamil *u*

- ஜ + ி → ஜு *ju*
- ஜ + ஁ → ஜூ *jū*

Ligatures with ra. Based on typographical preferences, the consonant ra ர may change shape to ர, when it ligates. Such change, if it occurs, will happen only when the ர form of U+0BB0 ர TAMIL LETTER RA would not be confused with the nominal form ர of U+0BBE TAMIL VOWEL SIGN AA (namely, when ர is combined with ஃ, ி, or ீ). This change in shape is illustrated in Figure 9-20.

Figure 9-20. Tamil Ligatures with *ra*

$$\begin{aligned} \text{ர} + \text{◌} &\rightarrow \text{ṙ} \text{ } r \\ \text{ர} + \text{◌ி} &\rightarrow \text{ṙi} \text{ } ri \\ \text{ர} + \text{◌ீ} &\rightarrow \text{ṙī} \text{ } rī \end{aligned}$$

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 9-20*.

Ligatures with aa in Traditional Tamil Orthography. In traditional Tamil orthography, the vowel sign *aa* ◌ஶ optionally ligates with ண, ண, or ற, as illustrated in *Figure 9-21*.

Figure 9-21. Traditional Tamil Ligatures with *aa*

$$\begin{aligned} \text{ண} + \text{◌ஶ} &\rightarrow \text{ṇā} \text{ } ṇā \\ \text{ண} + \text{◌ஶ} &\rightarrow \text{ṇā} \text{ } ṇā \\ \text{ற} + \text{◌ஶ} &\rightarrow \text{ṙā} \text{ } rā \end{aligned}$$

These ligations also affect the right-hand part of two-part vowels, as shown in *Figure 9-22*.

Figure 9-22. Traditional Tamil Ligatures with *o*

$$\begin{aligned} \text{ண} + \text{◌ொ} &\rightarrow \text{ṇō} \text{ } ṇō \\ \text{ண} + \text{◌ோ} &\rightarrow \text{ṇō} \text{ } ṇō \\ \text{ன} + \text{◌ொ} &\rightarrow \text{ṇō} \text{ } ṇō \\ \text{ன} + \text{◌ோ} &\rightarrow \text{ṇō} \text{ } ṇō \\ \text{ற} + \text{◌ொ} &\rightarrow \text{ṙō} \text{ } rō \\ \text{ற} + \text{◌ோ} &\rightarrow \text{ṙō} \text{ } rō \end{aligned}$$

Ligatures with ai in Traditional Tamil Orthography. In traditional Tamil orthography, the left-side vowel sign *ai* ◌ஶ is also subject to a change in form. It is rendered as ௪ when it occurs on the left side of ண, ண, ஸ, or ஶ, as illustrated in *Figure 9-23*.

Figure 9-23. Traditional Tamil Ligatures with *ai*

ண + ீ → ணை *nai*
 ன + ீ → னை *nai*
 ல + ீ → லை *lai*
 ள + ீ → ளை *lai*

By contrast, in modern Tamil orthography, this vowel does not change its shape, as shown in Figure 9-24.

Figure 9-24. Vowel *ai* in Modern Tamil

ண + ீ → ணை *nai*

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 9.1, *Devanagari*.

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 9-22 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 9-22, 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 9-22. Tamil Vowels, Consonants, and Syllables

	A	AA	I	II	U	UU	E	EE	AI	O	OO	AU	
	ஃ 0B83	அ 0B85	ஆ 0B86	இ 0B87	ஈ 0B88	உ 0B89	ஊ 0B8A	எ 0B8E	ஏ 0B8F	ஐ 0B90	ஓ 0B92	ஔ 0B93	ஔள 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
V	வ் 0BB5 0BCD	வ 0BB5	வா 0BB5 0BBE	வி 0BB5 0BBF	வீ 0BB5 0BC0	வு 0BB5 0BC1	வூ 0BB5 0BC2	வெ 0BB5 0BC6	வே 0BB5 0BC7	வை 0BB5 0BC8	வொ 0BB5 0BCA	வோ 0BB5 0BCB	வௌ 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

		A	AA	I	II	U	UU	E	EE	AI	O	OO	AU
J	జ్	జ	జా	జి	జీ	జూ	జౌ	జె	జే	జై	జొ	జో	జేణ
	0B9C 0BCD	0B9C	0B9C 0BBE	0B9C 0BBF	0B9C 0BC0	0B9C 0BC1	0B9C 0BC2	0B9C 0BC6	0B9C 0BC7	0B9C 0BC8	0B9C 0BCA	0B9C 0BCB	0B9C 0BCC
SH	ష	ష	షా	షి	షీ	షూ	షౌ	షె	షే	షై	షొ	షో	షేణ
	0BB6 0BCD	0BB6	0BB6 0BBE	0BB6 0BBF	0BB6 0BC0	0BB6 0BC1	0BB6 0BC2	0BB6 0BC6	0BB6 0BC7	0BB6 0BC8	0BB6 0BCA	0BB6 0BCB	0BB6 0BCC
SS	ష్	ష్	షా	షి	షీ	షూ	షౌ	షె	షే	షై	షొ	షో	షేణ
	0BB7 0BCD	0BB7	0BB7 0BBE	0BB7 0BBF	0BB7 0BC0	0BB7 0BC1	0BB7 0BC2	0BB7 0BC6	0BB7 0BC7	0BB7 0BC8	0BB7 0BCA	0BB7 0BCB	0BB7 0BCC
S	స్	స్	సా	సి	సీ	సూ	సౌ	సె	సే	సై	సొ	సో	సేణ
	0BB8 0BCD	0BB8	0BB8 0BBE	0BB8 0BBF	0BB8 0BC0	0BB8 0BC1	0BB8 0BC2	0BB8 0BC6	0BB8 0BC7	0BB8 0BC8	0BB8 0BCA	0BB8 0BCB	0BB8 0BCC
H	హ	హ	హా	హి	హీ	హూ	హౌ	హె	హే	హై	హొ	హో	హేణ
	0BB9 0BCD	0BB9	0BB9 0BBE	0BB9 0BBF	0BB9 0BC0	0BB9 0BC1	0BB9 0BC2	0BB9 0BC6	0BB9 0BC7	0BB9 0BC8	0BB9 0BCA	0BB9 0BCB	0BB9 0BCC
KSS	క	క	కా	కి	కీ	కూ	కౌ	కె	కే	కై	కొ	కో	కేణ
	0B95 0BCD 0BB7 0BCD	0B95 0BCD 0BB7	0B95 0BCD 0BB7 0BBE	0B95 0BCD 0BB7 0BBF	0B95 0BCD 0BB7 0BC0	0B95 0BCD 0BB7 0BC1	0B95 0BCD 0BB7 0BC2	0B95 0BCD 0BB7 0BC6	0B95 0BCD 0BB7 0BC7	0B95 0BCD 0BB7 0BC8	0B95 0BCD 0BB7 0BCA	0B95 0BCD 0BB7 0BCB	0B95 0BCD 0BB7 0BCC

SHRII	ఱ
	0BB6 0BCD 0BB0 0BC0

9.7 Telugu

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, Orissa, Madhya Pradesh, and West Bengal. The Telugu script became distinct by the thirteenth century CE and shares ancestors with the Kannada script.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. Table 9-23 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 9-23. Telugu Vowel Letters

For	Use	Do Not Use
ఱ	0C13	<0C12, 0C55>
ఱ	0C14	<0C12, 0C4C>
ఱ	0C40	<0C3F, 0C55>
ఱ	0C47	<0C46, 0C55>
ఱ	0C4B	<0C4A, 0C55>

Rendering Behavior. Telugu script rendering is similar to that of other Brahmic scripts in the Unicode Standard—in particular, the Tamil script. Unlike Tamil, however, the Telugu

script writes conjunct characters with subscript letters. 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 script. When a virama (called *virāmamu* in Telugu) or certain vowel signs are added to a letter with this headstroke, it is replaced:

$$U+0C15 \text{ ka} + U+0C4D \text{ virama} + U+200C \text{ ZERO WIDTH NON-JOINER} \rightarrow \text{ᳵ} (k)$$

$$U+0C15 \text{ ka} + U+0C3F \text{ vowel sign } i \rightarrow \text{ᳶ} (ki)$$

Telugu consonant clusters are most commonly represented by a subscripted, and often transformed, consonant glyph for the second element of the cluster:

$$U+0C17 \text{ ga} + U+0C4D \text{ virama} + U+0C17 \text{ ga} \rightarrow \text{᳷} (gga)$$

$$U+0C15 \text{ ka} + U+0C4D \text{ virama} + U+0C15 \text{ ka} \rightarrow \text{᳸} (kka)$$

$$U+0C15 \text{ ka} + U+0C4D \text{ virama} + U+0C2F \text{ ya} \rightarrow \text{᳹} (kya)$$

$$U+0C15 \text{ ka} + U+0C4D \text{ virama} + U+0C37 \text{ ssa} \rightarrow \text{ᳺ} (kṣa)$$

Nakāra-Pollu. The sequence <U+0C28 TELUGU LETTER NA, U+0C4D TELUGU SIGN VIRAMA> can have two representations in Telugu text. The first is the “regular” or “new style” form ᳵ, which takes its shape from the glyphs in the sequence <U+0C28 ᳵ TELUGU LETTER NA, U+0C4D ᳵ TELUGU SIGN VIRAMA>. Older texts display the other vowel-less form ᳶ, called *nakāra-pollu*. The two forms are semantically identical. Fonts should render the sequence <U+0C28 TELUGU LETTER NA, U+0C4D TELUGU SIGN VIRAMA> with either the old-style glyph ᳶ or the new style glyph ᳵ. The character U+200C ZERO WIDTH NON-JOINER can be used to prevent interaction of this sequence with following consonants, as shown in Table 9-24.

Table 9-24. Rendering of Telugu *na + virama*

Font	Sequence	Rendering
Old Style	<i>na + virama</i>	ᳶ
	<i>na + virama</i> + U+200C ZERO WIDTH NON-JOINER + <i>da</i>	ᳶ᳚
New Style	<i>na + virama</i>	ᳵ
	<i>na + virama</i> + U+200C ZERO WIDTH NON-JOINER + <i>da</i>	ᳵ᳚
All Fonts	<i>na + virama + da</i>	ᳶ᳚

Reph. In modern Telugu, U+0C30 TELUGU LETTER RA 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:

$$U+0C30 \text{ ra} + U+0C4D \text{ virama} + U+0C2E \text{ ma} \rightarrow \text{᳼} (rma)$$


However, in older texts, U+0C30 TELUGU LETTER RA takes the reduced (or *reph*) form ᳶ when it appears first in a consonant cluster, and the following consonant maintains its nominal form:

$$U+0C30 \text{ ra} + U+0C4D \text{ virama} + U+0C2E \text{ ma} \rightarrow \text{ᳶ᳚} (rma)$$

U+200D ZERO WIDTH JOINER is placed immediately after the *virama* to render the *reph* explicitly in modern texts:

$$U+0C30 \text{ ra} + U+0C4D \text{ virama} + U+200D \text{ ZERO WIDTH JOINER} + U+0C2E \text{ ma} \rightarrow \text{ᳶ᳚}$$

To prevent display of a *reph*, U+200D ZERO WIDTH JOINER is placed after the *ra*, but preceding the *virama*:

U+0C30 ರ ra + U+200D  ZWJ + U+0C4D ಿ virama + U+0C2E ಳ
ma → ರೳ

Special Characters. U+0C55 TELUGU LENGTH MARK is provided as an encoding for the second element of the vowel U+0C47 TELUGU VOWEL SIGN EE. U+0C56 TELUGU AI LENGTH MARK is provided as an encoding for the second element of the surroundant 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 9.6, Tamil*.

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 *halli*, 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 9.1, Devanagari*.

9.8 Kannada

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 9.1, Devanagari*, for further information.

The Unicode Standard follows the ISCII layout for encoding, which also reflects the traditional Kannada alphabetic order.

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 vertical stem replaced by the halant sign, which marks a character core. The stem glyph is graphically and historically related to the sign denoting the inherent /a/ vowel, U+0C85 ಅ KANNADA LETTER A. 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 9-25 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 9-25. Kannada Vowel Letters

For	Use	Do Not Use
ಊ	0C8A	<0C89, 0CBE>
ಋ	0C94	<0C92, 0CCC>
ಋ	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 ligatures 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 ೀ KANNADA LENGTH MARK is provided as an encoding for the right side of the two-part vowel U+0CC7 ು KANNADA VOWEL SIGN EE should it be necessary for processing. Likewise, U+0CD6 ು KANNADA AI LENGTH MARK is provided as an encoding for the right side of the two-part vowel U+0CC8 ೂ 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 ಃ KANNADA LETTER FA is actually an obsolete Kannada letter that is transliterated in Dravidian scholarship as *z*, *l*, or *r*. This form should have been named “LLLA”, rather than “FA”, so the name in this standard is simply a mistake. This letter has not been actively used in Kannada since the end of the tenth century. Collations should treat U+0CDE as following U+0CB3 KANNADA LETTER LLA.

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. Unlike in Devanagari and some other Indian scripts, all of the dependent vowels in Kannada are depicted to the right of their consonant letters. Hence there is no need to reorder the elements in mapping from the logical (character) store to the presentation (glyph) rendering, and vice versa.

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 ZERO WIDTH NON-JOINER is introduced immediately after the encoded dead consonant that is to be excluded from conjunct formation. See *Section 9.1, Devanagari*, for examples.

Consonant Clusters Involving RA. Whenever a consonant cluster is formed with the U+0CB0 ರ KANNADA LETTER RA 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* + zw + U+0CCD ೆ halant + U+0C95 ಕ *ka* → ರ್ಕ *rka*

U+0C95 ಕ *ka* + U+0CCD ೆ halant + U+0CB0 ರ *ra* → ಕ್ರ *kra*

Modifier Mark Rules. In addition to the vowel signs, one 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 called 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 9.1, Devanagari*.

9.9 Malayalam

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.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. *Table 9-26* 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.

Rendering Behavior. The shapes of Malayalam letters closely resemble those of Tamil. Malayalam, however, has a very full and complex set of conjunct consonant forms. In the

Table 9-26. Malayalam Vowel Letters

For	Use	Do Not Use
ഈ	0D08	<0D07, 0D57>
ഉ	0D0A	<0D09, 0D57>
ഐ	0D10	<0D0E, 0D46>
ഓ	0D13	<0D12, 0D3E>
ഔ	0D14	<0D12, 0D57>

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 9-27 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 9-27. Malayalam Orthographic Reform

Syllable		Older Orthography	Reformed Orthography
<i>ku</i>	ക + ു	ക	കു
<i>gu</i>	ഗ + ു	ഗ	ഗു
<i>chu</i>	ച + ു	ച	ചു
<i>ju</i>	ജ + ു	ജ	ജു
<i>nu</i>	ന + ു	ന	നു
<i>tu</i>	ത + ു	ത	തു
<i>nu</i>	ന + ു	ന	നു
<i>bhu</i>	ഭ + ു	ഭ	ഭു
<i>ru</i>	ര + ു	ര	രു
<i>śu</i>	ശ + ു	ശ	ശു
<i>hu</i>	ഹ + ു	ഹ	ഹു
<i>kū</i>	ക + ൂ	ക	കു
<i>gū</i>	ഗ + ൂ	ഗ	ഗു
<i>chū</i>	ച + ൂ	ച	ചു
<i>jū</i>	ജ + ൂ	ജ	ജു
<i>nū</i>	ന + ൂ	ന	നു
<i>tū</i>	ത + ൂ	ത	തു
<i>nū</i>	ന + ൂ	ന	നു
<i>bhū</i>	ഭ + ൂ	ഭ	ഭു
<i>rū</i>	ര + ൂ	ര	രു
<i>śū</i>	ശ + ൂ	ശ	ശു
<i>hū</i>	ഹ + ൂ	ഹ	ഹു

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

candrakala in Malayalam. Table 9-28 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 9-28. Malayalam Conjuncts

ക	+	്	+	ഷ	→	കഷ	(kṣa)
ക	+	്	+	ക	→	കക	(kka)
ജ	+	്	+	ഞ	→	ജഞ	(jña)
ട	+	്	+	ട	→	ട്ട	(ṭṭa)
പ	+	്	+	പ	→	പ്പ	(ppa)
ച	+	്	+	ര	→	ചര	(ccha)
ബ	+	്	+	ബ	→	ബ്ബ	(bba)
ന	+	്	+	യ	→	ന്യ	(nya)
പ	+	്	+	ര	→	പ്ര	(pra)
ര	+	്	+	പ	→	രപ	(rpa)
ശ	+	്	+	വ	→	ശവ	(śva)

When the *candrakala* sign is visibly shown in Malayalam, it usually indicates the suppression of the inherent vowel, but it sometimes indicates instead a reduced schwa sound [ə], often called “half-u” or *samvruthokaram*. In the later case, there can also be a -u vowel sign, and the base character can be a vowel letter. In all cases, the *candrakala* 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. Examples are shown in Table 9-29.

Table 9-29. Candrakala Examples

പാലു്	/paalə/ milk	0D2A, 0D3E, 0D32, 0D41, 0D4D
എന്നാ	/ənnaa/ on which day?	0D0E, 0D4D, 0D28, 0D4D, 0D28, 0D3E
ഐശീല്	/aishiiləm/ than ice	0D10, 0D36, 0D40, 0D32, 0D4D, 0D02

The *anusvara* can be seen after after vowel letters, as in ഇൗൗൗൗ <0D08, 0D02, 0D02, 0D02, 0D02>. Vowel signs can also be seen after digits, as in 355ൗ <0033, 0035, 0035, 0D3E, 0D02>. More generally, rendering engines should be prepared to handle Malayalam letters (including vowel letters), digits (both European and Malayalam), dashes, 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.

Chillu Characters. The six *chillu* or *cillakṣaram* characters, U+0D7A..U+0D7F, encode dead consonants (those without an inherent vowel). To simplify the discussion here, the formal names of the characters are shortened to use the terms that are typically used in spoken discussion of the *chillu* characters: *chillu-n* for MALAYALAM LETTER CHILLU N, and so forth.

In Malayalam-language text, *chillu* characters never start a word. The *chillu* letters *-nn*, *-n*, *-rr*, *-l*, and *-ll* are quite common; *chillu-k* is not very common.

Prior to Unicode Version 5.1, the representation of text with *chillus* was problematic, and not clearly described in the text of the standard. Because older data will use different representation for *chillus*, implementations must be prepared to handle both kinds of data. For the *chillu* letters considered in isolation, *Table 9-30* shows the relation between their representation in Unicode Version 5.0 and earlier, and the recommended representation starting with Unicode Version 5.1.

Table 9-30. Atomic Encoding of Malayalam *Chillus*

Visual	Representation in 5.0 and Prior	Preferred 5.1 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
ൿ	<i>undefined</i>	0D7F MALAYALAM LETTER CHILLU K

Special Cases Involving ra. There are a number of textual representation and reading issues involving the letter *ra*. These issues are discussed here and tables of explicit examples are presented.

The letter *ra* is normally read /r/. Repetition of that sound is written by two occurrences of the letter: *ra*. Each occurrence can bear a vowel sign.

Repetition of the letter, written either *ra* or *ra*, is also used for the sound /tt/. The sequence of two *ra* 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 *ra* in text is ambiguous in reading. The reader must generally use the context to understand if this is read /rr/ or /tt/. It is only when a vowel part appears between the two *ra* that the reading is unambiguously /rr/. 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 *ra*, regardless of the reading of that text. The sequence <0D31, 0D4D, 0D31> is rendered as *ra*. In both cases, vowel signs can be used as appropriate, as shown in *Table 9-31*.

Table 9-31. Malayalam /rr/ and /tt/

പാറ്റ	0D2A 0D3E 0D31 0D31	/paatta/	cockroach
പാറ്റ	0D2A 0D3E 0D31 0D4D 0D31		
മാറ്റൊലി	0D2E 0D3E 0D31 0D46 0D31 0D3E 0D32 0D3F	/maattoli/	echo
മാറ്റൊലി	0D2E 0D3E 0D31 0D4D 0D31 0D46 0D3E 0D32 0D3F		

Table 9-31. Malayalam /rr/ and /tt/ (Continued)

ബാറ്ററി	0D2C 0D3E 0D31 0D31 0D31 0D3F	/baattari/	battery
ബാറ്ററി	0D2C 0D3E 0D31 0D4D 0D31 0D31 0D3F		
സൂററ്	0D38 0D42 0D31 0D31 0D31 0D4D	/suurratt/	(name of a place)
സൂററ്	0D38 0D42 0D31 0D31 0D4D 0D31 0D4D		
ടെമ്പററി	0D1F 0D46 0D02 0D2A 0D31 0D31 0D3F	/temparrari/	temporary (English loan word)
ലക്ചററോട്	0D32 0D46 0D15 0D4D 0D1A 0D31 0D31 0D4B 0D1F 0D4D	/lekcararoot/	to the lecturer

A very similar situation exists for the combination of റ്റ chillu-*n* and റാ *ra*. When used side by side, റ്ററ can be read either /nr/ or /nt/, while റ്റ is always read /nt/.

The sequence <0D7B, 0D31> is rendered as റ്ററ, regardless of the reading of that text. The sequence <0D7B, 0D4D, 0D31> is rendered as റ്റ. In both cases, vowel signs can be used as appropriate, as shown in Table 9-32.

Table 9-32. Malayalam /nr/ and /nt/

ആന്റോ	0D06 0D7B 0D47 0D31 0D3E	/aantoo/	(proper name)
ആന്റോ	0D06 0D7B 0D4D 0D31 0D47 0D3E		
എൻറോൾ	0D0E 0D7B 0D31 0D47 0D3E 0D7A	/enrool/	enroll (English word)

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 over the consonant following it. Conceptually, the *dot reph* is analogous to the sequence <*ra*, *virama*>, but when followed by another consonant, the Malayalam cluster <*ra*, *virama*, C2> normally assumes the C2 conjoining form. U+0D4E MALAYALAM LETTER DOT REPH occurs first, in logical order, even though it displays as a dot above the succeeding consonant. It has the character properties of a letter, and is not considered a combining mark.

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 CHILLU RR.

The Malayalam *dot reph* was in common print usage until 1970, but has fallen into disuse. Words that formerly used *dot reph* are now spelled using U+0D7C MALAYALAM CHILLU RR or the respective C2-conjoining forms. The *dot reph* form is predominantly used by those who completed elementary education in Malayalam prior to 1970.

Historic Characters. The four characters, *avagraha*, *vocalic rr sign*, *vocalic l sign*, and *vocalic ll sign*, are only used to write Sanskrit words in the Malayalam script. The *avagraha* is the most common of the four, followed by the *vocalic l sign*. There are six characters used for the archaic number system, including characters for numbers 10, 100, 1000 and fractions. The *date mark* is used 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.

Special 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 “ഌ” on the left side and “ഏ” 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 above.

For a detailed discussion of the use of two-part vowels, see “Two-Part Vowels” in *Section 9.6, Tamil*.

Punctuation. Danda and double danda marks as well as some other unified punctuation used with Malayalam are found in the Devanagari block; see *Section 9.1, Devanagari*.

Chapter 10

South Asian Scripts-II

This chapter documents scripts of South Asia aside from the major official scripts of India, which are described in *Chapter 9, South Asian Scripts-I*.

The following South Asian scripts are described in this chapter:

<i>Sinhala</i>	<i>Kaithi</i>	<i>Meetei Mayek</i>
<i>Tibetan</i>	<i>Saurashtra</i>	<i>Ol Chiki</i>
<i>Lepcha</i>	<i>Sharada</i>	<i>Sora Sompeng</i>
<i>Phags-pa</i>	<i>Takri</i>	<i>Kharoshthi</i>
<i>Limbu</i>	<i>Chakma</i>	<i>Brahmi</i>
<i>Syloti Nagri</i>		

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.

Sinhala is an official script of Sri Lanka, where it is used to write the majority language, also known as Sinhala.

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.

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.

Phags-pa is a historical script related to Tibetan that was created as the national script of the Mongol empire. Even though Phags-pa was used mostly in Eastern and Central Asia for writing text in the Mongolian and Chinese languages, it is discussed in this chapter because of its close historical connection to the Tibetan script.

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.

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.

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.

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. There are efforts to revive its use for Dogri and other languages.

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.

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.

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 Orissa.

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.

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.

10.1 Sinhala

***Sinhala:* U+0D80–U+0DFF**

The Sinhala script, also known as Sinhalese, 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, අඳ <U+0D85, U+0DAC> *añḍa* [aⁿḍa] “sound” versus අඳව <U+0D85, U+0DAB, U+0DCA, U+0DA9> *aṇḍa* [aṇḍa] “egg.” In addition, Sinhala has separate distinct signs for both a short and a long low front vowel sounding similar to the initial vowel of the English word “apple,” usually represented in IPA as U+00E6 æ LATIN SMALL LETTER AE (*ash*). The independent forms of these

vowels are encoded at U+0D87 and U+0D88; the corresponding dependent forms are U+0DD0 and U+0DD1.

Because of these extra letters, the encoding for Sinhala does not precisely follow the pattern established for the other Indic scripts (for example, Devanagari). It does use the same general structure, making use of phonetic order, matra reordering, and use of the virama (U+0DCA SINHALA SIGN AL-LAKUNA) to indicate conjunct consonant clusters. Sinhala does not use half-forms in the Devanagari manner, but does use many ligatures.

Vowel Letters. Vowel letters are encoded atomically in Unicode, even if they can be analyzed visually as consisting of multiple parts. Table 10-1 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 10-1. Sinhala Vowel Letters

To Represent	Use	Do Not Use
ඒ	0D86	<0D85, 0DCF>
ඒෆ	0D87	<0D85, 0DD0>
ඒෆ්	0D88	<0D85, 0DD1>
ඊ	0D8C	<0D8B, 0DDF>
ඊෆෆ	0D8E	<0D8D, 0DD8>
ඊෆ්	0D90	<0D8F, 0DDF>
ඊෆ්	0D92	<0D91, 0DCA>
ඊෆ්	0D93	<0D91, 0DD9>
ඊෆ්	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 SINHALA LETTER RAYANNA and U+0DB1 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.

Historical Symbols. Neither U+0DF4 SINHALA PUNCTUATION KUNDDALIYA nor the Sinhala numerals are in general use today, having been replaced by Western-style punctuation and Western digits. The *kunddaliya* was formerly used as a full stop or period. It is included for scholarly use. The Sinhala numerals are not presently encoded.

10.2 Tibetan

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 Warty 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.

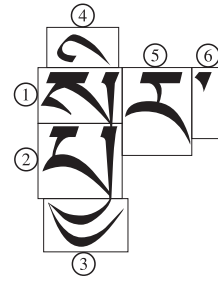
Each word in Tibetan has a base or root consonant. The base consonant can be written singly or it can have other consonants added above or below it to make a vertically "stacked" letter. Tibetan grammar contains a very complete set of rules regarding letter gender, and these rules dictate which letters 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 [tʃi:], is a typical example of a Tibetan syllable that includes a stack comprising a head letter, two subscript letters, and a vowel sign. *Figure 10-1* shows the characters in the order in which they appear in the backing store.

The model adopted to encode the Tibetan lettering set described above contains the following groups of items: Tibetan consonants, vowels, numerals, punctuation, ornamental signs

Figure 10-1. 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



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. This is called *ra-go* (ra-mgo). 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 *wazur* (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 subjoined 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 character 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-mgo* 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+0FB E and U+0FC F.

U+0F34, which means “et cetera” or “and so on,” is used after the first few *tsek-bar* of a recurring phrase. U+0FB E (often three times) indicates a refrain.

U+0F36 and U+0FB F 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, Bengali) have similar fractional concepts. The value of each half-number is 0.5 less than the number within which it appears. These forms are used only in some traditional contexts and appear as the *last* digit of a multidigit number. For example, the sequence of digits “U+0F24 U+0F2C” represents the number 42.5 or forty-two and one-half.

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 included in normal Tibetan. They could be made using the method described earlier for Tibetan stacked consonants, generally by subjoining “ha”. However, to maintain consistency in transliterated texts and for ease in transmission and searching, it is recommended that implementations of Sanskrit in the Tibetan script use the precomposed forms of aspirated letters (and U+0F69, “ka + reversed sha”) whenever possible, rather than implementing these consonants as completely decomposed stacks. Implementations must ensure that decomposed stacks and precomposed forms are interpreted equivalently (see Section 3.7, *Decomposition*). The compound consonants are

explicitly coded as follows: U+0F93 TIBETAN SUBJOINED LETTER GHA, U+0F9D TIBETAN SUBJOINED LETTER DDHA, U+0FA2 TIBETAN SUBJOINED LETTER DHA, U+0FA7 TIBETAN SUBJOINED LETTER BHA, U+0FAC TIBETAN SUBJOINED LETTER DZHA, and U+0FB9 TIBETAN SUBJOINED LETTER KSSA.

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 (*ngaro*) is the anusvara. See *Section 9.1, 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 10-2* is not usually padded as in the second line of *Figure 10-2*, 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 10-2* 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 10-2* is not acceptable.

Figure 10-2. Justifying Tibetan Tseks

```

1 འགྲུག།
2 འགྲུག།.....
3 འགྲུག། .....
4 འགྲུག། .....

```

Tibetan text has two rules regarding the formatting of 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 *go-yig* (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” *go-yig* (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 with 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.

10.3 Lepcha

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 Phyagrdor 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 counter-clockwise, 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 letter forms, not clearly related to the corresponding consonants without *-la*.

Retroflex Consonants. The Lepcha language contains three retroflex consonants: [ʈ], [ʈʰ], and [ɖ]. 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 LEP-

CHA 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 10-2.

Table 10-2. Lepcha Syllabic Structure

Class	Example	Encoding
consonant, letter a	ᱠ	[U+1C00..U+1C23, U+1C4D..U+1C4F]
nukta	ᱡ	U+1C37
medial -ra	ᱢ	U+1C25
medial -ya	ᱣ	U+1C24
dependent vowel	ᱤ	[U+1C26..U+1C2C]
final consonant sign	ᱥ	[U+1C2D..U+1C35]
syllabic modifier	ᱦ	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 only after the inherent vowel *-a* or the dependent vowels *-aa* and *-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 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.

10.4 Phags-pa

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 Uighur-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 Uighur, 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 *candrabinu* (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.

Candrabinu. 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 *candrabinu* followed by the letter *o*. To simplify cursor placement, text selection, and so on, the *candrabinu* 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 *Figure 10-3*.

Figure 10-3. 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 10-3*.

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

Table 10-3. Phags-pa Positional Forms of I, U, E, and O

Letter	Isolate	Initial	Medial	Final
U+A85E PHAGS-PA LETTER I	ᠶ	ᠶ	ᠶ	ᠶ
U+A85F PHAGS-PA LETTER U	ᠸ	ᠸ	ᠸ	ᠸ
U+A860 PHAGS-PA LETTER E	ᠺ	ᠺ	ᠺ	ᠺ
U+A861 PHAGS-PA LETTER O	ᠻ	ᠻ	ᠻ	ᠻ

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 16, 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 the letter *o*

<U+200D, U+A861> selects the final form of the letter *o*

<U+A861, U+200D> selects the initial form of the letter *o*

Conversely, ZWNJ may be used to inhibit expected shaping. For example, the sequence <U+A85E, U+200C, U+A85F, U+200C, U+A860, U+200C, U+A861> selects the isolate forms of the letters *i*, *u*, *e*, and *o*.

Mirrored Variants. The four characters U+A869 PHAGS-PA LETTER TTA, U+A86A PHAGS-PA LETTER TTTHA, 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 10-4*.

Table 10-4. Contextual Glyph Mirroring in Phags-pa

Character	Syllables with Glyph Mirroring	Syllables without Glyph Mirroring
U+A856 PHAGS-PA LETTER SMALL A	<i>tthā</i>	<i>ttā, tthā</i>
U+A85E PHAGS-PA LETTER I	<i>tthi, nni</i>	<i>tthi</i>
U+A85F PHAGS-PA LETTER U	<i>nnu</i>	
U+A860 PHAGS-PA LETTER E	<i>tthe, dde, nne</i>	
U+A85C PHAGS-PA LETTER HA	<i>ddha</i>	
U+A868 PHAGS-PA SUBJOINED LETTER YA	<i>nnya</i>	

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 10-5. Note that only the variation sequences shown in Table 10-5 are valid; any other sequence of a Phags-pa letter and VS1 is unspecified.

Table 10-5. Phags-pa Standardized Variants

Character Sequence	Description of Variant Appearance
<U+A856, U+FE00>	<i>phags-pa letter reversed shaping small a</i>
<U+A85C, U+FE00>	<i>phags-pa letter reversed shaping ha</i>
<U+A85E, U+FE00>	<i>phags-pa letter reversed shaping i</i>
<U+A85F, U+FE00>	<i>phags-pa letter reversed shaping u</i>
<U+A860, U+FE00>	<i>phags-pa letter reversed shaping e</i>
<U+A868, U+FE00>	<i>phags-pa letter reversed shaping ya</i>

In Table 10-5, “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 Figure 10-4.

Figure 10-4. 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 *tthi*. 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>.

10.5 Limbu

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 *yak-thuy*. 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 ཨ -ya, ས -ra, and སྐ -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 𑄛 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 𑄀 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 𑄀 [kɔ], 𑄁 [kʰɔ], and so on, and do not represent the Limbu syllables 𑄀 [ka], 𑄁 [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 𑄀𑄀 [ɔm].

Vowel Length. Vowel length is phonologically distinctive in many contexts. Length in open syllables is indicated by writing U+193A 𑄛 LIMBU SIGN KEMPHRENG, which looks like the diaeresis sign, over the initial consonant or cluster: 𑄛 𑄀 *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: 𑄛 𑄀 *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*: 𑄛 𑄀 *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: 𑄛 𑄀 *nesse* “it lay,” 𑄛 𑄀 *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 𑄛 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 𑄀 *na* is placed before 𑄀 *ta*, and the letter 𑄀 *gha* is placed at the end of the alphabet.

Glyph Placement. The glyph positions for Limbu combining characters are summarized in Table 10-6.

Punctuation. The main punctuation mark used is the double vertical line, U+0965 DEVANAGARI DOUBLE DANDA. U+1945 𑄛 LIMBU QUESTION MARK and U+1944 𑄛 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

Table 10-6. Positions of Limbu Combining Characters

Syllable	Glyphs	Code Point Sequence
ta	ᳵ	190B 1920
ti	ᳶ	190B 1921
tu	᳷	190B 1922
tee	᳸	190B 1923
tai	᳹	190B 1924
too	ᳺ	190B 1925
tau	᳻	190B 1926
te	᳼	190B 1927
to	᳽	190B 1928
tya	᳾	190B 1929
tra	᳿	190B 192A
twa	᳠	190B 192B
tak	᳡	U+190B U+1930
taŋ	᳢	U+190B U+1931
tañ	᳣	U+190B U+1932
tat	᳤	U+190B U+1933
tan	᳥	U+190B U+1934
tap	᳦	U+190B U+1935
tam	᳧	U+190B U+1936
tar	᳨	U+190B U+1937
tal	ᳩ	U+190B U+1938
tā	ᳪ	U+190B U+1920 U+193A
tī	ᳫ	U+190B U+1921 U+193A

in the same documents. U+1940 ᳬ LIMBU SIGN LOO is used for the exclamatory particle *lo*. This particle is also often simply spelled out ᳭.

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.

10.6 Syloti Nagri

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 𑒠 SYLOTI NAGRI VOWEL SIGN I and U+A826 𑒡 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. Syloti Nagri is atypical of Indic scripts in use of the *virama* (*hasanta*) and conjuncts. Conjuncts are not strictly correlated with the phonology being represented. They are neither necessary in contexts involving a dead consonant, nor are they limited to such contexts. *Hasanta* was only recently introduced into the script and is used only in limited contexts. Conjuncts are not limited to sequences involving dead consonants but can be formed from pairs of characters of almost any type (consonant, independent vowel, dependent vowel) and can represent a wide variety of syllables. It is generally unnecessary to overtly indicate dead consonants with a conjunct or explicit *hasanta*. The only restriction is that an overtly rendered *hasanta* cannot occur in connection with the first element of a conjunct. The absence of *hasanta* does not imply a live consonant and has no bearing on the occurrence of conjuncts. Similarly, the absence of a conjunct does not imply a live consonant and has no bearing on the occurrence of *hasanta*.

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.

10.7 Kaithi

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 nineteenth 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 9.1, 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 character that interacts with digits, occurring either above or below a digit. The position of the KAITHI NUMBER SIGN indicates its usage: when the mark occurs above a digit, it indicates a number in an itemized list, similar to U+2116 NUMERO SIGN. If it occurs below a digit, it indicates a numerical reference. Like U+0600 ARABIC NUMBER SIGN and the other Arabic signs that span numbers (see *Section 8.2, Arabic*), the KAITHI NUMBER SIGN precedes the numbers they graphically interact with, rather than following them, as would combining characters. The U+110BC KAITHI ENUMERATION SIGN is the spacing version of the KAITHI NUMBER SIGN, and is used 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 marks of punctuation 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..096F should be employed. To indicate fractions and unit marks, Kaithi makes use of the numbers encoded in the Common Indic Number Forms block, U+A830..A839.

10.8 Saurashtra

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, Saurash-

tra 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+A8CF SAURASHTRA SIGN VIRAMA. An exception to the non-occurrence of complex consonant clusters is the conjunct *ksa*, 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 *ksa*, 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.

10.9 Sharada

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 9.1, Devanagari*.

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 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 𑆃 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 following voiceless velar stops:

U+111C2 𑆂 jihvamuliya + U+11191 𑆑 ka → 𑆒

U+111C2 𑆂 jihvamuliya + U+11192 𑆒 kha → 𑆓

Upadhmaniya is used to represent the bilabial fricative [ɸ] in the context of following voiceless labial stops:

U+111C3 𑆃 upadhmaniya + U+111A5 𑆕 pa → 𑆔

U+111C3 𑆃 upadhmaniya + U+111A6 𑆖 pha → 𑆕

Punctuation. U+111C7 𑆇 SHARADA ABBREVIATION SIGN appears after letters or combinations of letters. It marks the sequence as an abbreviation. A word separator, U+111C8 𑆈 SHARADA SEPARATOR, indicates word and other boundaries. Sharada also makes use of two script-specific dandas: U+111C5 𑆅 SHARADA DANDA and U+111C6 𑆆 SHARADA DOUBLE DANDA.

Digits. Sharada has a distinctive set of digits encoded in the range U+111D0..U+111D9.

10.10 Takri

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 Devāṣeṣa, which emerged in the 14th century. Devāṣeṣa 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 Devāṣeṣa 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 letter forms, 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 10-7* 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 10-7. Takri Vowel Letters

For	Use	Do Not Use
𑆚	11681	<11680, 116AD>
𑆛	11687	<11686, 116B2>
𑆜	11688	<11680, 116B4>
𑆝	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).

10.11 Chakma

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* (*cānaphupudā*) 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 in the script: 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. Chakma uses the *maayyaa* (killer) to invoke conjoined consonants. Most letters have their vowels killed with the use of the explicit *maayyaa* character. In addition to the visible killer, there is an explicit conjunct-forming character (*virama*), permitting the user to choose between the subjoining style and the ligating style. Whether a conjunct is required or not is part of the spelling of a word.

In principle, nothing prevents the visible killer from appearing together with a subjoining sequence formed with *virama*. However, in practice, combinations of *virama* and *maayyaa* following a consonant are not meaningful, as both kill the inherent vowel.

In 2001, an orthographic reform was recommended in the book *Cānmā 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.



10.12 Meetei Mayek

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  may be read *kara* or *kra*,  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.

Meetei Mayak Extensions: U+AAE0–U+AAF6

The Meetei Mayak 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+AAF5 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 Brahmic-derived scripts. For example, the conjunct /ñha/ is represented by the sequence <ABC9, AAF5, 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 REPRESENTATION MARK.

10.13 Ol Chiki

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 Orissan Mayurbhañj district. The script has received some official recognition by the Orissan 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 alphabetic and has none of the structural properties of the abugidas typical for other Indic scripts. There are separate letters representing consonants and vowels. A number of modifier letters are used to indicate tone, nasalization, vowel length, and deglottalization. There are no combining characters in the script.

Ol Chiki is written from left to right.

Digits. The Ol Chiki script has its own set of digits. These are separately encoded in the Ol Chiki block.

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.

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+1C5E, 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-GAHLAA 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 [kʰ] 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.

10.14 Sora Sompeng

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 Orissa-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 structured as 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.

Linebreaking. Letters and digits behave as in Latin and other alphabetic scripts.

10.15 Kharoshthi

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 nineteenth 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 10-5*). 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.

The exact details of the origin of the Kharoshthi script remain obscure, but it is almost certainly related to Aramaic. The Kharoshthi 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 documents 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.

Figure 10-5. Geographical Extent of the Kharoshthi Script



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.

Diacritic Marks and Vowels. All vowels other than *a* are written with diacritic 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 10-6*. The numerals are encoded in the range U+10A40..U+10A47.

Figure 10-6. Kharoshthi Number 1996

𑖑𑖒𑖓𑖔𑖕𑖖𑖗𑖘𑖙𑖚𑖛𑖜𑖝𑖞𑖟𑖠𑖡𑖢𑖣𑖤𑖥𑖦𑖧𑖨𑖩𑖪𑖫𑖬𑖭𑖮𑖯𑖰𑖱𑖲𑖳𑖴𑖵𑖶𑖷𑖸𑖹𑖺𑖻𑖼𑖽𑖾𑗀𑖿𑗁𑗂𑗃𑗄𑗅𑗆𑗇𑗈𑗉𑗊𑗋𑗌𑗍𑗎𑗏𑗐𑗑𑗒𑗓𑗔𑗕𑗖𑗗𑗘𑗙𑗚𑗛𑗜𑗝𑗞𑗟𑗠𑗡𑗢𑗣𑗤𑗥𑗦𑗧𑗨𑗩𑗪𑗫𑗬𑗭𑗮𑗯𑗰𑗱𑗲𑗳𑗴𑗵𑗶𑗷𑗸𑗹𑗺𑗻𑗼𑗽𑗾𑗿𑘀𑘁𑘂𑘃𑘄𑘅𑘆𑘇𑘈𑘉𑘊𑘋𑘌𑘍𑘎𑘏𑘐𑘑𑘒𑘓𑘔𑘕𑘖𑘗𑘘𑘙𑘚𑘛𑘜𑘝𑘞𑘟𑘠𑘡𑘢𑘣𑘤𑘥𑘦𑘧𑘨𑘩𑘪𑘫𑘬𑘭𑘮𑘯𑘰𑘱𑘲𑘳𑘴𑘵𑘶𑘷𑘸𑘹𑘺𑘻𑘼𑘽𑘾𑘿𑙀𑙁𑙂𑙃𑙄𑙅𑙆𑙇𑙈𑙉𑙊𑙋𑙌𑙍𑙎𑙏𑙐𑙑𑙒𑙓𑙔𑙕𑙖𑙗𑙘𑙙𑙚𑙛𑙜𑙝𑙞𑙟𑙠𑙡𑙢𑙣𑙤𑙥𑙦𑙧𑙨𑙩𑙪𑙫𑙬𑙭𑙮𑙯𑙰𑙱𑙲𑙳𑙴𑙵𑙶𑙷𑙸𑙹𑙺𑙻𑙼𑙽𑙾𑙿𑚀𑚁𑚂𑚃𑚄𑚅𑚆𑚇𑚈𑚉𑚊𑚋𑚌𑚍𑚎𑚏𑚐𑚑𑚒𑚓𑚔𑚕𑚖𑚗𑚘𑚙𑚚𑚛𑚜𑚝𑚞𑚟𑚠𑚡𑚢𑚣𑚤𑚥𑚦𑚧𑚨𑚩𑚪𑚫𑚬𑚭𑚮𑚯𑚰𑚱𑚲𑚳𑚴𑚵𑚷𑚶𑚸𑚹𑚺𑚻𑚼𑚽𑚾𑚿𑛀𑛁𑛂𑛃𑛄𑛅𑛆𑛇𑛈𑛉𑛊𑛋𑛌𑛍𑛎𑛏𑛐𑛑𑛒𑛓𑛔𑛕𑛖𑛗𑛘𑛙𑛚𑛛𑛜𑛝𑛞𑛟𑛠𑛡𑛢𑛣𑛤𑛥𑛦𑛧𑛨𑛩𑛪𑛫𑛬𑛭𑛮𑛯𑛰𑛱𑛲𑛳𑛴𑛵𑛶𑛷𑛸𑛹𑛺𑛻𑛼𑛽𑛾𑛿𑜀𑜁𑜂𑜃𑜄𑜅𑜆𑜇𑜈𑜉𑜊𑜋𑜌𑜍𑜎𑜏𑜐𑜑𑜒𑜓𑜔𑜕𑜖𑜗𑜘𑜙𑜚𑜛𑜜𑜝𑜞𑜟𑜠𑜡𑜢𑜣𑜤𑜥𑜦𑜧𑜨𑜩𑜪𑜫𑜬𑜭𑜮𑜯𑜰𑜱𑜲𑜳𑜴𑜵𑜶𑜷𑜸𑜹𑜺𑜻𑜼𑜽𑜾𑜿𑝀𑝁𑝂𑝃𑝄𑝅𑝆𑝇𑝈𑝉𑝊𑝋𑝌𑝍𑝎𑝏𑝐𑝑𑝒𑝓𑝔𑝕𑝖𑝗𑝘𑝙𑝚𑝛𑝜𑝝𑝞𑝟𑝠𑝡𑝢𑝣𑝤𑝥𑝦𑝧𑝨𑝩𑝪𑝫𑝬𑝭𑝮𑝯𑝰𑝱𑝲𑝳𑝴𑝵𑝶𑝷𑝸𑝹𑝺𑝻𑝼𑝽𑝾𑝿𑞀𑞁𑞂𑞃𑞄𑞅𑞆𑞇𑞈𑞉𑞊𑞋𑞌𑞍𑞎𑞏𑞐𑞑𑞒𑞓𑞔𑞕𑞖𑞗𑞘𑞙𑞚𑞛𑞜𑞝𑞞𑞟𑞠𑞡𑞢𑞣𑞤𑞥𑞦𑞧𑞨𑞩𑞪𑞫𑞬𑞭𑞮𑞯𑞰𑞱𑞲𑞳𑞴𑞵𑞶𑞷𑞸𑞹𑞺𑞻𑞼𑞽𑞾𑞿𑟀𑟁𑟂𑟃𑟄𑟅𑟆𑟇𑟈𑟉𑟊𑟋𑟌𑟍𑟎𑟏𑟐𑟑𑟒𑟓𑟔𑟕𑟖𑟗𑟘𑟙𑟚𑟛𑟜𑟝𑟞𑟟𑟠𑟡𑟢𑟣𑟤𑟥𑟦𑟧𑟨𑟩𑟪𑟫𑟬𑟭𑟮𑟯𑟰𑟱𑟲𑟳𑟴𑟵𑟶𑟷𑟸𑟹𑟺𑟻𑟼𑟽𑟾𑟿𑠀𑠁𑠂𑠃𑠄𑠅𑠆𑠇𑠈𑠉𑠊𑠋𑠌𑠍𑠎𑠏𑠐𑠑𑠒𑠓𑠔𑠕𑠖𑠗𑠘𑠙𑠚𑠛𑠜𑠝𑠞𑠟𑠠𑠡𑠢𑠣𑠤𑠥𑠦𑠧𑠨𑠩𑠪𑠫𑠬𑠭𑠮𑠯𑠰𑠱𑠲𑠳𑠴𑠵𑠶𑠷𑠸𑠺𑠹𑠻𑠼𑠽𑠾𑠿𑡀𑡁𑡂𑡃𑡄𑡅𑡆𑡇𑡈𑡉𑡊𑡋𑡌𑡍𑡎𑡏𑡐𑡑𑡒𑡓𑡔𑡕𑡖𑡗𑡘𑡙𑡚𑡛𑡜𑡝𑡞𑡟𑡠𑡡𑡢𑡣𑡤𑡥𑡦𑡧𑡨𑡩𑡪𑡫𑡬𑡭𑡮𑡯𑡰𑡱𑡲𑡳𑡴𑡵𑡶𑡷𑡸𑡹𑡺𑡻𑡼𑡽𑡾𑡿𑢀𑢁𑢂𑢃𑢄𑢅𑢆𑢇𑢈𑢉𑢊𑢋𑢌𑢍𑢎𑢏𑢐𑢑𑢒𑢓𑢔𑢕𑢖𑢗𑢘𑢙𑢚𑢛𑢜𑢝𑢞𑢟𑢠𑢡𑢢𑢣𑢤𑢥𑢦𑢧𑢨𑢩𑢪𑢫𑢬𑢭𑢮𑢯𑢰𑢱𑢲𑢳𑢴𑢵𑢶𑢷𑢸𑢹𑢺𑢻𑢼𑢽𑢾𑢿𑣀𑣁𑣂𑣃𑣄𑣅𑣆𑣇𑣈𑣉𑣊𑣋𑣌𑣍𑣎𑣏𑣐𑣑𑣒𑣓𑣔𑣕𑣖𑣗𑣘𑣙𑣚𑣛𑣜𑣝𑣞𑣟𑣠𑣡𑣢𑣣𑣤𑣥𑣦𑣧𑣨𑣩𑣪𑣫𑣬𑣭𑣮𑣯𑣰𑣱𑣲𑣳𑣴𑣵𑣶𑣷𑣸𑣹𑣺𑣻𑣼𑣽𑣾𑣿𑤀𑤁𑤂𑤃𑤄𑤅𑤆𑤇𑤈𑤉𑤊𑤋𑤌𑤍𑤎𑤏𑤐𑤑𑤒𑤓𑤔𑤕𑤖𑤗𑤘𑤙𑤚𑤛𑤜𑤝𑤞𑤟𑤠𑤡𑤢𑤣𑤤𑤥𑤦𑤧𑤨𑤩𑤪𑤫𑤬𑤭𑤮𑤯𑤰𑤱𑤲𑤳𑤴𑤵𑤶𑤷𑤸𑤹𑤺𑤻𑤼𑤽𑤾𑤿𑥀𑥁𑥂𑥃𑥄𑥅𑥆𑥇𑥈𑥉𑥊𑥋𑥌𑥍𑥎𑥏𑥐𑥑𑥒𑥓𑥔𑥕𑥖𑥗𑥘𑥙𑥚𑥛𑥜𑥝𑥞𑥟𑥠𑥡𑥢𑥣𑥤𑥥𑥦𑥧𑥨𑥩𑥪𑥫𑥬𑥭𑥮𑥯𑥰𑥱𑥲𑥳𑥴𑥵𑥶𑥷𑥸𑥹𑥺𑥻𑥼𑥽𑥾𑥿𑦀𑦁𑦂𑦃𑦄𑦅𑦆𑦇𑦈𑦉𑦊𑦋𑦌𑦍𑦎𑦏𑦐𑦑𑦒𑦓𑦔𑦕𑦖𑦗𑦘𑦙𑦚𑦛𑦜𑦝𑦞𑦟𑦠𑦡𑦢𑦣𑦤𑦥𑦦𑦧𑦨𑦩𑦪𑦫𑦬𑦭𑦮𑦯𑦰𑦱𑦲𑦳𑦴𑦵𑦶𑦷𑦸𑦹𑦺𑦻𑦼𑦽𑦾𑦿𑧀𑧁𑧂𑧃𑧄𑧅𑧆𑧇𑧈𑧉𑧊𑧋𑧌𑧍𑧎𑧏𑧐𑧑𑧒𑧓𑧔𑧕𑧖𑧗𑧘𑧙𑧚𑧛𑧜𑧝𑧞𑧟𑧠𑧡𑧢𑧣𑧤𑧥𑧦𑧧𑧨𑧩𑧪𑧫𑧬𑧭𑧮𑧯𑧰𑧱𑧲𑧳𑧴𑧵𑧶𑧷𑧸𑧹𑧺𑧻𑧼𑧽𑧾𑧿𑨀𑨁𑨂𑨃𑨄𑨅𑨆𑨇𑨈𑨉𑨊𑨋𑨌𑨍𑨎𑨏𑨐𑨑𑨒𑨓𑨔𑨕𑨖𑨗𑨘𑨙𑨚𑨛𑨜𑨝𑨞𑨟𑨠𑨡𑨢𑨣𑨤𑨥𑨦𑨧𑨨𑨩𑨪𑨫𑨬𑨭𑨮𑨯𑨰𑨱𑨲𑨳𑨴𑨵𑨶𑨷𑨸𑨹𑨺𑨻𑨼𑨽𑨾𑨿𑩀𑩁𑩂𑩃𑩄𑩅𑩆𑩇𑩈𑩉𑩊𑩋𑩌𑩍𑩎𑩏𑩐𑩑𑩒𑩓𑩔𑩕𑩖𑩗𑩘𑩙𑩚𑩛𑩜𑩝𑩞𑩟𑩠𑩡𑩢𑩣𑩤𑩥𑩦𑩧𑩨𑩩𑩪𑩫𑩬𑩭𑩮𑩯𑩰𑩱𑩲𑩳𑩴𑩵𑩶𑩷𑩸𑩹𑩺𑩻𑩼𑩽𑩾𑩿𑪀𑪁𑪂𑪃𑪄𑪅𑪆𑪇𑪈𑪉𑪊𑪋𑪌𑪍𑪎𑪏𑪐𑪑𑪒𑪓𑪔𑪕𑪖𑪗𑪘𑪙𑪚𑪛𑪜𑪝𑪞𑪟𑪠𑪡𑪢𑪣𑪤𑪥𑪦𑪧𑪨𑪩𑪪𑪫𑪬𑪭𑪮𑪯𑪰𑪱𑪲𑪳𑪴𑪵𑪶𑪷𑪸𑪹𑪺𑪻𑪼𑪽𑪾𑪿𑫀𑫁𑫂𑫃𑫄𑫅𑫆𑫇𑫈𑫉𑫊𑫋𑫌𑫍𑫎𑫏𑫐𑫑𑫒𑫓𑫔𑫕𑫖𑫗𑫘𑫙𑫚𑫛𑫜𑫝𑫞𑫟𑫠𑫡𑫢𑫣𑫤𑫥𑫦𑫧𑫨𑫩𑫪𑫫𑫬𑫭𑫮𑫯𑫰𑫱𑫲𑫳𑫴𑫵𑫶𑫷𑫸𑫹𑫺𑫻𑫼𑫽𑫾𑫿𑬀𑬁𑬂𑬃𑬄𑬅𑬆𑬇𑬈𑬉𑬊𑬋𑬌𑬍𑬎𑬏𑬐𑬑𑬒𑬓𑬔𑬕𑬖𑬗𑬘𑬙𑬚𑬛𑬜𑬝𑬞𑬟𑬠𑬡𑬢𑬣𑬤𑬥𑬦𑬧𑬨𑬩𑬪𑬫𑬬𑬭𑬮𑬯𑬰𑬱𑬲𑬳𑬴𑬵𑬶𑬷𑬸𑬹𑬺𑬻𑬼𑬽𑬾𑬿𑭀𑭁𑭂𑭃𑭄𑭅𑭆𑭇𑭈𑭉𑭊𑭋𑭌𑭍𑭎𑭏𑭐𑭑𑭒𑭓𑭔𑭕𑭖𑭗𑭘𑭙𑭚𑭛𑭜𑭝𑭞𑭟𑭠𑭡𑭢𑭣𑭤𑭥𑭦𑭧𑭨𑭩𑭪𑭫𑭬𑭭𑭮𑭯𑭰𑭱𑭲𑭳𑭴𑭵𑭶𑭷𑭸𑭹𑭺𑭻𑭼𑭽𑭾𑭿𑮀𑮁𑮂𑮃𑮄𑮅𑮆𑮇𑮈𑮉𑮊𑮋𑮌𑮍𑮎𑮏𑮐𑮑𑮒𑮓𑮔𑮕𑮖𑮗𑮘𑮙𑮚𑮛𑮜𑮝𑮞𑮟𑮠𑮡𑮢𑮣𑮤𑮥𑮦𑮧𑮨𑮩𑮪𑮫𑮬𑮭𑮮𑮯𑮰𑮱𑮲𑮳𑮴𑮵𑮶𑮷𑮸𑮹𑮺𑮻𑮼𑮽𑮾𑮿𑯀𑯁𑯂𑯃𑯄𑯅𑯆𑯇𑯈𑯉𑯊𑯋𑯌𑯍𑯎𑯏𑯐𑯑𑯒𑯓𑯔𑯕𑯖𑯗𑯘𑯙𑯚𑯛𑯜𑯝𑯞𑯟𑯠𑯡𑯢𑯣𑯤𑯥𑯦𑯧𑯨𑯩𑯪𑯫𑯬𑯭𑯮𑯯𑯰𑯱𑯲𑯳𑯴𑯵𑯶𑯷𑯸𑯹𑯺𑯻𑯼𑯽𑯾𑯿𑰀𑰁𑰂𑰃𑰄𑰅𑰆𑰇𑰈𑰉𑰊𑰋𑰌𑰍𑰎𑰏𑰐𑰑𑰒𑰓𑰔𑰕𑰖𑰗𑰘𑰙𑰚𑰛𑰜𑰝𑰞𑰟𑰠𑰡𑰢𑰣𑰤𑰥𑰦𑰧𑰨𑰩𑰪𑰫𑰬𑰭𑰮𑰯𑰰𑰱𑰲𑰳𑰴𑰵𑰶𑰷𑰸𑰹𑰺𑰻𑰼𑰽𑰾𑰿𑱀𑱁𑱂𑱃𑱄𑱅𑱆𑱇𑱈𑱉𑱊𑱋𑱌𑱍𑱎𑱏𑱐𑱑𑱒𑱓𑱔𑱕𑱖𑱗𑱘𑱙𑱚𑱛𑱜𑱝𑱞𑱟𑱠𑱡𑱢𑱣𑱤𑱥𑱦𑱧𑱨𑱩𑱪𑱫𑱬𑱭𑱮𑱯𑱰𑱱𑱲𑱳𑱴𑱵𑱶𑱷𑱸𑱹𑱺𑱻𑱼𑱽𑱾𑱿𑲀𑲁𑲂𑲃𑲄𑲅𑲆𑲇𑲈𑲉𑲊𑲋𑲌𑲍𑲎𑲏𑲐𑲑𑲒𑲓𑲔𑲕𑲖𑲗𑲘𑲙𑲚𑲛𑲜𑲝𑲞𑲟𑲠𑲡𑲢𑲣𑲤𑲥𑲦𑲧𑲨𑲩𑲪𑲫𑲬𑲭𑲮𑲯𑲰𑲱𑲲𑲳𑲴𑲵𑲶𑲷𑲸𑲹𑲺𑲻𑲼𑲽𑲾𑲿𑳀𑳁𑳂𑳃𑳄𑳅𑳆𑳇𑳈𑳉𑳊𑳋𑳌𑳍𑳎𑳏𑳐𑳑𑳒𑳓𑳔𑳕𑳖𑳗𑳘𑳙𑳚𑳛𑳜𑳝𑳞𑳟𑳠𑳡𑳢𑳣𑳤𑳥𑳦𑳧𑳨𑳩𑳪𑳫𑳬𑳭𑳮𑳯𑳰𑳱𑳲𑳳𑳴𑳵𑳶𑳷𑳸𑳹𑳺𑳻𑳼𑳽𑳾𑳿𑴀𑴁𑴂𑴃𑴄𑴅𑴆𑴇𑴈𑴉𑴊𑴋𑴌𑴍𑴎𑴏𑴐𑴑𑴒𑴓𑴔𑴕𑴖𑴗𑴘𑴙𑴚𑴛𑴜𑴝𑴞𑴟𑴠𑴡𑴢𑴣𑴤𑴥𑴦𑴧𑴨𑴩𑴪𑴫𑴬𑴭𑴮𑴯𑴰𑴱𑴲𑴳𑴴𑴵𑴶𑴷𑴸𑴹𑴺𑴻𑴼𑴽𑴾𑴿𑵀𑵁𑵂𑵃𑵄𑵅𑵆𑵇𑵈𑵉𑵊𑵋𑵌𑵍𑵎𑵏𑵐𑵑𑵒𑵓𑵔𑵕𑵖𑵗𑵘𑵙𑵚𑵛𑵜𑵝𑵞𑵟𑵠𑵡𑵢𑵣𑵤𑵥𑵦𑵧𑵨𑵩𑵪𑵫𑵬𑵭𑵮𑵯𑵰𑵱𑵲𑵳𑵴𑵵𑵶𑵷𑵸𑵹𑵺𑵻𑵼𑵽𑵾𑵿𑶀𑶁𑶂𑶃𑶄𑶅𑶆𑶇𑶈𑶉𑶊𑶋𑶌𑶍𑶎𑶏𑶐𑶑𑶒𑶓𑶔𑶕𑶖𑶗𑶘𑶙𑶚𑶛𑶜𑶝𑶞𑶟𑶠𑶡𑶢𑶣𑶤𑶥𑶦𑶧𑶨𑶩𑶪𑶫𑶬𑶭𑶮𑶯𑶰𑶱𑶲𑶳𑶴𑶵𑶶𑶷𑶸𑶹𑶺𑶻𑶼𑶽𑶾𑶿𑷀𑷁𑷂𑷃𑷄𑷅𑷆𑷇𑷈𑷉𑷊𑷋𑷌𑷍𑷎𑷏𑷐𑷑𑷒𑷓𑷔𑷕𑷖𑷗𑷘𑷙𑷚𑷛𑷜𑷝𑷞𑷟𑷠𑷡𑷢𑷣𑷤𑷥𑷦𑷧𑷨𑷩𑷪𑷫𑷬𑷭𑷮𑷯𑷰𑷱𑷲𑷳𑷴𑷵𑷶𑷷𑷸𑷹𑷺𑷻𑷼𑷽𑷾𑷿𑸀𑸁𑸂𑸃𑸄𑸅𑸆𑸇𑸈𑸉𑸊𑸋𑸌𑸍𑸎𑸏𑸐𑸑𑸒𑸓𑸔𑸕𑸖𑸗𑸘𑸙𑸚𑸛𑸜𑸝𑸞𑸟𑸠𑸡𑸢𑸣𑸤𑸥𑸦𑸧𑸨𑸩𑸪𑸫𑸬𑸭𑸮𑸯𑸰𑸱𑸲𑸳𑸴𑸵𑸶𑸷𑸸𑸹𑸺𑸻𑸼𑸽𑸾𑸿𑹀𑹁𑹂𑹃𑹄𑹅𑹆𑹇𑹈𑹉𑹊𑹋𑹌𑹍𑹎𑹏𑹐𑹑𑹒𑹓𑹔𑹕𑹖𑹗𑹘𑹙𑹚𑹛𑹜𑹝𑹞𑹟𑹠𑹡𑹢𑹣𑹤𑹥𑹦𑹧𑹨𑹩𑹪𑹫𑹬𑹭𑹮𑹯𑹰𑹱𑹲𑹳𑹴𑹵𑹶𑹷𑹸𑹹𑹺𑹻𑹼𑹽𑹾𑹿𑺀𑺁𑺂𑺃𑺄𑺅𑺆𑺇𑺈𑺉𑺊𑺋𑺌𑺍𑺎𑺏𑺐𑺑𑺒𑺓𑺔𑺕𑺖𑺗𑺘𑺙𑺚𑺛𑺜𑺝𑺞𑺟𑺠𑺡𑺢𑺣𑺤𑺥𑺦𑺧𑺨𑺩𑺪𑺫𑺬𑺭𑺮𑺯𑺰𑺱𑺲𑺳𑺴𑺵𑺶𑺷𑺸𑺹𑺺𑺻𑺼𑺽𑺾𑺿𑻀𑻁𑻂𑻃𑻄𑻅𑻆𑻇𑻈𑻉𑻊𑻋𑻌𑻍𑻎𑻏𑻐𑻑𑻒𑻓𑻔𑻕𑻖𑻗𑻘𑻙𑻚𑻛𑻜𑻝𑻞𑻟𑻠𑻡𑻢𑻣𑻤𑻥𑻦𑻧𑻨𑻩𑻪𑻫𑻬𑻭𑻮𑻯𑻰𑻱𑻲𑻳𑻴𑻵𑻶𑻷𑻸𑻹𑻺𑻻𑻼𑻽𑻾𑻿𑼀𑼁𑼂𑼃𑼄𑼅𑼆𑼇𑼈𑼉𑼊𑼋𑼌𑼍𑼎𑼏𑼐𑼑𑼒𑼓𑼔𑼕𑼖𑼗𑼘𑼙𑼚𑼛𑼜𑼝𑼞𑼟𑼠𑼡𑼢𑼣𑼤𑼥𑼦𑼧𑼨𑼩𑼪𑼫𑼬𑼭𑼮𑼯𑼰𑼱𑼲𑼳𑼴𑼵𑼶𑼷𑼸𑼹𑼺𑼻𑼼𑼽𑼾𑼿𑽀𑽁𑽂𑽃𑽄𑽅𑽆𑽇𑽈𑽉𑽊𑽋𑽌𑽍𑽎𑽏𑽐𑽑𑽒𑽓𑽔𑽕𑽖𑽗𑽘𑽙𑽚𑽛𑽜𑽝𑽞𑽟𑽠𑽡𑽢𑽣𑽤𑽥𑽦𑽧𑽨𑽩𑽪𑽫𑽬𑽭𑽮𑽯𑽰𑽱𑽲𑽳𑽴𑽵𑽶𑽷𑽸𑽹𑽺𑽻𑽼𑽽𑽾𑽿𑾀𑾁𑾂𑾃𑾄𑾅𑾆𑾇𑾈𑾉𑾊𑾋𑾌𑾍𑾎𑾏𑾐𑾑𑾒𑾓𑾔𑾕𑾖𑾗𑾘𑾙𑾚𑾛𑾜𑾝𑾞𑾟𑾠𑾡𑾢𑾣𑾤𑾥𑾦𑾧𑾨𑾩𑾪𑾫𑾬𑾭𑾮𑾯𑾰𑾱𑾲𑾳𑾴𑾵𑾶𑾷𑾸𑾹𑾺𑾻𑾼𑾽𑾾𑾿𑿀𑿁𑿂𑿃𑿄𑿅𑿆𑿇𑿈𑿉𑿊𑿋𑿌𑿍𑿎𑿏𑿐𑿑𑿒𑿓𑿔𑿕𑿖𑿗𑿘𑿙𑿚𑿛𑿜𑿝𑿞𑿟𑿠𑿡𑿢𑿣𑿤𑿥𑿦𑿧𑿨𑿩𑿪𑿫𑿬𑿭𑿮𑿯𑿰𑿱𑿲𑿳𑿴𑿵𑿶𑿷𑿸𑿹𑿺𑿻𑿼𑿽𑿾𑿿𑀀𑀁𑀂𑀃𑀄𑀅𑀆𑀇𑀈𑀉𑀊𑀋𑀌𑀍𑀎𑀏𑀐𑀑𑀒𑀓𑀔𑀕𑀖𑀗𑀘𑀙𑀚𑀛𑀜𑀝𑀞𑀟𑀠𑀡𑀢𑀣𑀤𑀥𑀦𑀧𑀨𑀩𑀪𑀫𑀬𑀭𑀮𑀯𑀰𑀱𑀲𑀳𑀴𑀵𑀶𑀷𑀸𑀹𑀺𑀻𑀼𑀽𑀾𑀿𑁀𑁁𑁂𑁃𑁄𑁅𑁆𑁇𑁈𑁉𑁊𑁋𑁌𑁍𑁎𑁏𑁐𑁑𑁒𑁓𑁔𑁕𑁖𑁗𑁘𑁙𑁚𑁛𑁜𑁝𑁞𑁟𑁠𑁡𑁢𑁣𑁤𑁥𑁦𑁧𑁨𑁩𑁪𑁫𑁬𑁭𑁮𑁯𑁰𑁱𑁲𑁳𑁴𑁵𑁶𑁷𑁸𑁹𑁺𑁻𑁼𑁽𑁾𑁿𑂀𑂁𑂂𑂃𑂄𑂅𑂆𑂇𑂈𑂉𑂊𑂋𑂌𑂍𑂎𑂏𑂐𑂑𑂒𑂓𑂔𑂕𑂖𑂗𑂘𑂙𑂚𑂛𑂜𑂝𑂞𑂟𑂠𑂡𑂢𑂣𑂤𑂥𑂦𑂧𑂨𑂩𑂪𑂫𑂬𑂭𑂮𑂯𑂰𑂱𑂲𑂳𑂴𑂵𑂶𑂷𑂸𑂺𑂹𑂻𑂼𑂽𑂾𑂿𑃀𑃁𑃂𑃃𑃄𑃅𑃆𑃇𑃈𑃉𑃊𑃋𑃌𑃍𑃎𑃏𑃐𑃑𑃒𑃓𑃔𑃕𑃖𑃗𑃘𑃙𑃚𑃛𑃜𑃝𑃞𑃟𑃠𑃡𑃢𑃣𑃤𑃥𑃦𑃧𑃨𑃩𑃪𑃫𑃬𑃭𑃮𑃯𑃰𑃱𑃲𑃳𑃴𑃵𑃶𑃷𑃸𑃹𑃺𑃻𑃼𑃽𑃾𑃿𑄀𑄁𑄂𑄃𑄄𑄅𑄆𑄇𑄈𑄉𑄊𑄋𑄌𑄍𑄎𑄏𑄐𑄑𑄒𑄓𑄔𑄕𑄖𑄗𑄘𑄙𑄚𑄛𑄜𑄝𑄞𑄟𑄠𑄡𑄢𑄣𑄤𑄥𑄦𑄧𑄨𑄩𑄪𑄫

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 10-8*.

Table 10-8. Kharoshthi Vowel Signs

Type	Example	Group Members
Vowel sign i		
Horizontal	a + -i → i 𑀧 + 𑀭 → 𑀧𑀭	A, NA, HA
Vertical	tha + -i → thi 𑀧𑀭 + 𑀭 → 𑀧𑀭𑀭	THA, PA, PHA, MA, LA, SHA
Diagonal	ka + -i → ki 𑀧𑀭 + 𑀭 → 𑀧𑀭𑀭	All other letters
Vowel sign u		
Independent	ha + -u → hu 𑀧 + 𑀭 → 𑀧𑀭	TTA, HA
Ligated	ma + -u → mu 𑀭 + 𑀭 → 𑀭𑀭	MA
Attached	a + -u → u 𑀧 + 𑀭 → 𑀧𑀭	All other letters
Vowel sign vocalic r		
Attached	a + -r → r 𑀧 + 𑀭 → 𑀧𑀭	A, KA, KKA, KHA, GA, GHA, CA, CHA, JA, TA, DA, DHA, NA, PA, PHA, BA, BHA, VA, SHA, SA
Independent	ma + -r → mr 𑀭 + 𑀭 → 𑀭𑀭	MA, HA
Vowel sign e		
Horizontal	a + -e → e 𑀧 + 𑀭 → 𑀧𑀭	A, NA, HA
Vertical	tha + -e → the 𑀧𑀭 + 𑀭 → 𑀧𑀭𑀭	THA, PA, PHA, LA, SSA
Ligated	da + -e → de 𑀭 + 𑀭 → 𑀭𑀭	DA, MA
Diagonal	ka + -e → ke 𑀧𑀭 + 𑀭 → 𑀧𑀭𑀭	All other letters
Vowel sign o		
Vertical	pa + -o → po 𑀭 + 𑀭 → 𑀭𑀭	PA, PHA, YA, SHA
Diagonal	a + -o → o 𑀧 + 𑀭 → 𑀧𑀭	All other letters

Combining Vowel Modifiers. U+10A0C 𑀭 K HAROSH TH I VOWEL LENGTH MARK indicates equivalent long vowels and, when used in combination with -e and -o, indicates the diphthongs -ai and -au. U+10A0D 𑀭 K HAROSH TH I 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 𑀭 K HAROSH TH I SIGN ANUSVARA indicates nasalization, and

U+10A0F ̄̄ 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 10-9*.

Table 10-9. Kharoshthi Vowel Modifiers

Type	Example	Group Members
Vowel length mark	ma + ̄̄ → mā 𑀢 + ̄̄ → 𑀣	A, I, U, R, E, O
Double ring below	sa + ̣̣ → ṣ̣a 𑀤 + ̣̣ → 𑀥	A, U
Anusvara	a + -ṃ → aṃ 𑀦 + ̣ → 𑀧	A, I, U, R, E, O
Visarga	ka + -ḥ → kaḥ 𑀨 + ̄̄ → 𑀩	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 10-10*.

Table 10-10. Kharoshthi Consonant Modifiers

Type	Example	Group Members
Bar above	ja + ̄ → jā 𑀪 + ̄ → 𑀫	GA, CA, JA, NA, MA, SHA, SSA, SA, HA
Cauda	ga + ̣ → g̣a 𑀬 + ̣ → 𑀭	GA, JA, DDA, TA, DA, PA, YA, VA, SHA, SA
Dot below	ma + ̣ → ṃa 𑀮 + ̣ → 𑀯	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 10-11*.

Table 10-11. Examples of Kharoshthi Virama

Type	Example
Pure virama	$dha + i + k + \text{VIRAMA} \rightarrow dhik$ 𑀢 + 𑀣 + 𑀤 + 𑀥 → 𑀦
Ligatures	$ka + \text{VIRAMA} + sa \rightarrow ksa$ 𑀤 + 𑀥 + 𑀧 → 𑀨
Consonants with special combining forms	$sa + \text{VIRAMA} + ya \rightarrow sya$ 𑀧 + 𑀥 + 𑀩 → 𑀪
Consonants with full combined form	$ka + \text{VIRAMA} + ta \rightarrow kta$ 𑀤 + 𑀥 + 𑀫 → 𑀬

10.16 Brahmi

Brahmi: U+11000–U+1106F

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 10-12 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.

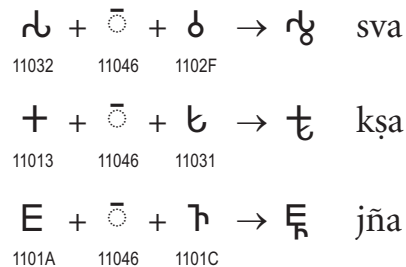
Table 10-12. Brahmi Vowel Letters

To Represent	Use	Do Not Use
𑀢	11006	<11005, 11038>
𑀣	1100C	<1100B, 1103E>
𑀤	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 10-8*. 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 *Figure 10-8*.

Figure 10-8. 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+10146 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/; that is, [x] and [f]. The velar and labial allophones of /h/, 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 Brahmi are covered by the Unicode encoding of Brahmi. For example, in one Tamil Brahmi system the inherent vowel of Brahmi consonant signs 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. Three consonant letters are encoded to represent sounds particular to Dravidian. These are U+11035 BRAHMI LETTER OLD TAMIL LLLA, U+11036 BRAHMI LETTER OLD TAMIL RRA, and U+11037 BRAHMI LETTER OLD TAMIL NNA.

Tamil Brahmi *pulli* (*virama*) had two functions: to cancel the inherent vowel of consonants; and to indicate the short vowels [e] and [o] in contrast to the long vowels [e:] and [o:] in Prakrit and Sanskrit. As a consequence, in Tamil Brahmi text, the *virama* is used not only after consonants, but also after the vowels *e* (U+1100E, U+11042) and *o* (U+11011, U+11044). This *pulli* is represented using U+11046 BRAHMI SIGN VIRAMA.

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.

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 decimal numbers that occurs side by side with the earlier numbering system in manuscripts and inscriptions during the late Brahmi period.

The set of additive-multiplicative numbers of the Brahmi script contains separate number signs for the digits from 1 to 9, the decades from 10 to 90, as well as signs for 100 and 1000. Numbers are written additively, with higher number signs preceding lower ones. Multiples of 100 and of 1000 are expressed multiplicatively, with the multiplier following and forming a ligature with 100 or 1000. There are examples from the middle and late Brahmi periods in which the signs for 200, 300, and 2000 appear in special forms and are 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 the modern decimal number system. Due to the different systemic features and shapes, the signs in this set have been encoding separately. These signs have the same properties as the modern Indian digits. Examples are shown in *Table 10-13*.

Table 10-13. Brahmi Positional Digits

Display	Value	Code Points
·	0	11066
↘	1	11067
२	2	11068
३	3	11069
४	4	1106A
↘·	10	<11067, 11066>
२३४	234	<11066, 11069, 1106A>

Chapter 11

Southeast Asian Scripts

This chapter documents the following scripts of Southeast Asia, Indonesia, and the Philippines:

<i>Thai</i>	<i>Tai Tham</i>	<i>Balinese</i>
<i>Lao</i>	<i>Tai Viet</i>	<i>Javanese</i>
<i>Myanmar</i>	<i>Kayah Li</i>	<i>Rejang</i>
<i>Khmer</i>	<i>Cham</i>	<i>Batak</i>
<i>Tai Le</i>	<i>Philippine scripts</i>	<i>Sundanese</i>
<i>New Tai Lue</i>	<i>Buginese</i>	

The scripts of Southeast Asia 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.

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.

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 Xishuang Banna 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 northwest-

ern 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.

There are four traditional Philippine scripts: Tagalog, Hanunóo, Buhid, and Tagbanwa. They have limited current use. They are discussed together, because each is structured quite similarly. Each is a very simplified abugida which makes use of two nonspacing vowel signs.

Although the official language of Indonesia, Bahasa Indonesia, is written in the Latin script, Indonesia has many local, traditional scripts, most of which are ultimately derived from Brahmi. Five of these scripts are documented in this chapter. Buginese is used for several different languages on the island of Sulawesi. Balinese and Javanese are closely related, highly ornate scripts; Balinese is 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.

11.1 Thai

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 11-1*.

Table 11-1. Glyph Positions in Thai Syllables

Syllable	Glyphs	Code Point Sequence
<i>ka</i>	กะ	0E01 0E30
<i>ka:</i>	กา	0E01 0E32
<i>ki</i>	กิ	0E01 0E34
<i>ki:</i>	กี	0E01 0E35
<i>ku</i>	กุ	0E01 0E38
<i>ku:</i>	กู	0E01 0E39
<i>ku'</i>	กิ	0E01 0E36
<i>ku':</i>	กี	0E01 0E37
<i>ke</i>	กะ	0E40 0E01 0E30
<i>ke:</i>	เก	0E40 0E01
<i>kae</i>	กะ	0E41 0E01 0E30
<i>kae:</i>	แก	0E41 0E01
<i>ko</i>	โกะ	0E42 0E01 0E30
<i>ko:</i>	โก	0E42 0E01
<i>ko'</i>	กะ	0E40 0E01 0E32 0E30
<i>ko':</i>	ก	0E01 0E2D
<i>koe</i>	กะ	0E40 0E01 0E2D 0E30
<i>koe:</i>	ก	0E40 0E01 0E2D
<i>kia</i>	กีย	0E40 0E01 0E35 0E22
<i>ku'a</i>	กือ	0E40 0E01 0E37 0E2D
<i>kua</i>	กัว	0E01 0E31 0E27
<i>kaw</i>	กา	0E40 0E01 0E32
<i>koe:y</i>	เกย	0E40 0E01 0E22
<i>kay</i>	ไก	0E44 0E01
<i>kay</i>	ไ	0E43 0E01
<i>kam</i>	กำ	0E01 0E33
<i>kri</i>	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 can grow to have a visible width when justified. See *Table 16-2*.

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.

11.2 Lao

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 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 11-2*.

Table 11-2. Glyph Positions in Lao Syllables

Syllable	Glyphs	Code Point Sequence
<i>ka</i>	ກະ	0E81 0EB0
<i>ka:</i>	ກາ	0E81 0EB2
<i>ki</i>	ກີ	0E81 0EB4
<i>ki:</i>	ກີ	0E81 0EB5
<i>ku</i>	ກຸ	0E81 0EB8
<i>ku:</i>	ກູ	0E81 0EB9
<i>ku'</i>	ກີ້	0E81 0EB6
<i>ku':</i>	ກີ້	0E81 0EB7
<i>ke</i>	ເກະ	0EC0 0E81 0EB0
<i>ke:</i>	ເກ	0EC0 0E81
<i>kae</i>	ແກະ	0EC1 0E81 0EB0
<i>kae:</i>	ແກ	0EC1 0E81
<i>ko</i>	ໂກະ	0EC2 0E81 0EB0
<i>ko:</i>	ໂກ	0EC2 0E81
<i>ko'</i>	ເກາະ	0EC0 0E81 0EB2 0EB0
<i>ko':</i>	ກໍ	0E81 0ECD
<i>koe</i>	ເກີ	0EC0 0E81 0EB4
<i>koe:</i>	ເກີ	0EC0 0E81 0EB5
<i>kia</i>	ເກີ້ວ ເກຢ	0EC0 0E81 0EB1 0EBD 0EC0 0E81 0EA2
<i>ku'a</i>	ເກີ້ອ	0EC0 0E81 0EB7 0EAD
<i>kua</i>	ກົວ	0E81 0EBB 0EA7
<i>kaw</i>	ເກົາ	0EC0 0E81 0EBB 0EB2
<i>koe:y</i>	ເກີ້ວ ເກຢ	0EC0 0E81 0EB5 0EBD 0EC0 0E81 0EB5 0EA2
<i>kay</i>	ໄກ	0EC4 0E81
<i>kay</i>	ໄກ	0EC3 0E81
<i>kam</i>	ກໍ່າ	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*.

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.

11.3 Myanmar

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 ◯ ka + U+1031 ◯ vowel sign e + U+102C ◯ vowel sign aa →
 ◯ ◯ ◯ /kàw/

U+1000 ◯ ka + U+1031 ◯ vowel sign e + U+102C ◯ vowel sign aa +
 U+103A ◯ asat → ◯ ◯ ◯ /kaw/

U+1000 ◯ ka + U+102D ◯ vowel sign i + U+102F ◯ vowel sign u → ◯ ◯
 /kol

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 C 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 ◯ MYANMAR LETTER KA would be written via the following sequence:

U+1004 C nga + U+103A ◯ asat + U+1039 ◯ virama + U+1000 ◯ ka
 → ◯ ◯ 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 ကွဲ [kjwei] (“to drop off”) would be written via the following sequence:

U+1000 က *ka* + U+103C ꠘ *medial ra* + U+103D ꠉ *medial wa* +
U+1031 ꠅ *vowel sign e* → ကွဲ /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 ꠙ 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 ꠇ 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 က *ya* + U+1031 ꠅ *vowel sign e* + U+102C ꠉ *vowel sign aa* +
U+1000 က *ka* + U+103A ꠇ *asat* + U+103B ꠘ *medial ya* + U+102C ꠉ
vowel sign aa + U+1038 ꠆ *visarga* → ကောတံပု : man, husband

U+1000 က *ka* + U+103B ꠘ *medial ya* + U+103D ꠉ *medial wa* +
U+1014 န *na* + U+103A ꠇ *asat* + U+102F ꠈ *vowel sign u* + U+1015 ပ *pa*
+ U+103A ꠇ *asat* → ကျွန်ုပ် I (first person singular)

Great sa. The *great sa* is encoded as U+103F ꠚ 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*, ဆဆ , and the sequence <U+101E, U+103A, U+101E> should be used for the form with an *asat sign*, ဆဆ .

Tall aa. The two shapes ꠉ and ꠉ 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 ꠉ MYANMAR VOWEL SIGN TALL AA and U+102C ꠉ 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 11-3.

Table 11-3. Myanmar Syllabic Structure

Class	Example	Encoding
<i>kinzi</i>	ꨀ	<U+1004, U+103A, U+1039>
<i>consonant and vowel letters</i>	ꨁ	[U+1000..U+102A, U+103F, U+104E]
<i>asat sign (for contractions)</i>	ꨂ	U+103A
<i>subscript consonant</i>	ꨃ	<U+1039, [U+1000..U+1019, U+101C, U+101E, U+1020, U+1021]>
<i>medial ya</i>	ꨄ	U+103B
<i>medial ra</i>	ꨅ	U+103C
<i>medial wa</i>	ꨆ	U+103D
<i>medial ha</i>	ꨇ	U+103E
<i>vowel sign e</i>	ꨈ	U+1031
<i>vowel sign i, ii, ai</i>	ꨉ, ꨊ, ꨋ	[U+102D, U+102E, U+1032]
<i>vowel sign u, uu</i>	ꨌ, ꨍ	[U+102F, U+1030]
<i>vowel sign tall aa, aa</i>	ꨎ, ꨏ	[U+102B, U+102C]
<i>anusvara</i>	ꨐ	U+1036
<i>asat sign</i>	ꨑ	U+103A
<i>dot below</i>	ꨒ	U+1037
<i>visarga</i>	ꨓ	U+1038

U+1031 ꨈ 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 11-3 nominally refers to the character sequences used in representing the syllabic structure of the Burmese language proper. It would require further extensions and modifications to cover the various other languages, such as Karen, Mon, and Shan, which also use the Myanmar script.

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).

Myanmar Extended-A: U+AA60–U+AA7F

This block provides additional characters to support Khamti Shan, Aiton and Phake. 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.

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 Khok 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 11-4*.

Table 11-4. 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	<i>unmarked</i>
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 MAIMAYOK.

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.

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+102E, 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.

11.4 Khmer

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 *akxaa 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 *akxaa crieng* (“slanted script”) and the *akxaa 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.

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 9.1, 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 11-5*, 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 11-5. Independent Khmer Vowel Characters

Name	Independent Vowel	Qa with Vowel Sign
<i>i</i>	ឺ	ឺ, ឺ, ឺ
<i>ii</i>	ឺ	ឺ, ឺ
<i>u</i>	្ហ	្ហ, ្ហ
<i>uk</i>	្ហ	្ហ
<i>uu</i>	្ហ	្ហ, ្ហ
<i>uuv</i>	្ហ	្ហ
<i>ry</i>	្ហ	្ហ
<i>ryy</i>	្ហ	្ហ

Table 11-5. Independent Khmer Vowel Characters (Continued)

Name	Independent Vowel	Qa with Vowel Sign
<i>ly</i>	ឺ	ឺ
<i>lyy</i>	ឺ	ឺ
<i>e</i>	ឯ	េ, ែ
<i>ai</i>	ឺ	ៃ
<i>oo</i>	ឺ, ឺ	ោ
<i>au</i>	ឺ	ោ

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:

លួ *lo* + *coeng* + *ngo* [lɲɔː] “sesame” (compare លង *lo* + *ngo* [lɔːŋ] “to haunt”)

លក្សី *lo* + *ka* + *coeng* + *sa* + *coeng* + *mo* + *ii* [ləksmei] “beauty, luck”

កាហ្វេ *ka* + *aa* + *ha* + *coeng* + *vo* + *e* [ka:feː] “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:

ទាំង *to* + *aa* + *nikahit* + *coeng* + *ngo* [tɛəŋ] “both” (= ទាំង) (≠ *ទាំង [tɲəəm])

ហើយ *ha* + *oe* + *coeng* + *yo* [haəi] “already” (= ហើយ) (≠ *ហើយ [hyaə])

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:

ឡើយ *qoo-1* + *coeng* + *yo* [ʔaoi] “to give” (= ឡើយ and also ឡើយ)

ឡើយ *qoo-1* + *coeng* + *mo* [ʔaom] “exclamation of solemn affirmation” (= ឡើយ)

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:

ផ្អែម *pha + coeng + qe + mo* [p^hʔaem] “sweet” (= ផ្អែម *pha + coeng + qa + ae + mo*)

ហ្វូទ័យ *ha + coeng + ry + to + samyok sannya + yo* [harutey] “heart”
(*royal*) (= ហ្វូទ័យ *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 *Table 11-6*. 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).

Table 11-6. Two Registers of Khmer Consonants

Row	First Register	Second Register
1	កំ <i>ka</i> [kɔː] “neck”	កំ <i>ko</i> [kɔː] “mute”
2	រំ <i>ro + muusikatoan</i> [rɔː] “small saw”	រំ <i>ro</i> [rɔː] “fence (in the water)”
3	សំកំ <i>sa + ka</i> [sɔːk] “to peel, to shed one’s skin”	សំកំ <i>sa + triisap + ka</i> [sɔːk] “to insert”
4	បំកំ <i>ba + ka</i> [bɔːk] “to return”	*បំកំ <i>ba + triisap + ka</i> [bɔːk]
5	បំមំ <i>ba + muusikatoan + mo</i> [pɔːm] “blockhouse”	ពំមំ <i>po + mo</i> [pɔːm] “to put into the mouth”
6	ក្អំ <i>ka + u + ro</i> [koː] “to stir”	ក្អំ <i>ko + u + ro</i> [kuː] “to sketch”

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 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 U+17D2 𑄲 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 11-7* 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 11-9*.

Table 11-7. 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
𑄶	17D2 1784	khmer consonant sign coeng ngo
𑄷	17D2 1785	khmer consonant sign coeng ca
𑄸	17D2 1786	khmer consonant sign coeng cha
𑄹	17D2 1787	khmer consonant sign coeng co
𑄺	17D2 1788	khmer consonant sign coeng cho
𑄻	17D2 1789	khmer consonant sign coeng nyo
𑄼	17D2 178A	khmer consonant sign coeng da
𑄽	17D2 178B	khmer consonant sign coeng ttha
𑄾	17D2 178C	khmer consonant sign coeng do
𑄿	17D2 178D	khmer consonant sign coeng ttho
𑅀	17D2 178E	khmer consonant sign coeng na
𑅁	17D2 178F	khmer consonant sign coeng ta
𑅂	17D2 1790	khmer consonant sign coeng tha
𑅃	17D2 1791	khmer consonant sign coeng to
𑅄	17D2 1792	khmer consonant sign coeng tho
𑅅	17D2 1793	khmer consonant sign coeng no
𑅆	17D2 1794	khmer consonant sign coeng ba
𑅇	17D2 1795	khmer consonant sign coeng pha
𑅈	17D2 1796	khmer consonant sign coeng po
𑅉	17D2 1797	khmer consonant sign coeng pho

Table 11-7. Khmer Subscript Consonant Signs (Continued)



Glyph	Code	Name
	17D2 1798	khmer consonant sign coeng mo
	17D2 1799	khmer consonant sign coeng yo
	17D2 179A	khmer consonant sign coeng ro
	17D2 179B	khmer consonant sign coeng lo
	17D2 179C	khmer consonant sign coeng vo
	17D2 179D	khmer consonant sign coeng sha
	17D2 179E	khmer consonant sign coeng ssa
	17D2 179F	khmer consonant sign coeng sa
	17D2 17A0	khmer consonant sign coeng ha
	17D2 17A1	khmer consonant sign coeng la
	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 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 [ǎm], not [a:m]. The sequences used for these combinations are shown in Table 11-8.

Table 11-8. 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:

ដំ *da + nikahit* [dɔm] “to pound” (compare ដំ *da + mo* [dɔ:m] “nec-tar”)

ព័ *po + aa + nikahit* [pɔəm] “to carry in the beak” (compare ព័ *po + aa + mo* [pɔəm] “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 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 11-9.

Table 11-9. Khmer Subscript Independent Vowel Signs

Glyph	Code	Name
្ក	17D2 17A7	khmer independent vowel sign coeng qu
្ខ	17D2 17AB	khmer independent vowel sign coeng ry
្គ	17D2 17AC	khmer independent vowel sign coeng ryy
្ឃ	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 without 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:

រាជ្ជី *ro + aa + co + ro + coeng + sa + ii* [rɛ̀ɛrsei] “king hermit”

អាឃី *qa + aa + yo + robat* [ʔa:rya] “civilized” (= អាឃ្យ *qa + aa + ro + coeng + yo*)

ព័ត៌មាន *po + ta + robat + mo + aa + no* [pɔ̀:ɔ̀mɛ̀ən] “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 *Figure 11-1*.

Figure 11-1. Common Ligatures in Khmer

ក ka + ា aa + រ ro = កា រ [ka:] “job”
 ប ba + ា aa = បា [ba:] “father, male of an animal”; used to prevent confusion with ហ ha
 ប ba + ោ au = បោ [baw] “to suck”
 ម mo + ួ coeng sa + ោ au = មួ ោ [msaw] “powder”
 ស sa + ង ngo + ួ coeng kha + ួ coeng yo + ា aa = សង្ឃួ ោ [sɔŋk^hya:] “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 KHMER LETTER NYO> is rendered according to context, as shown in *Figure 11-2*. 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 ឺ KHMER INDEPENDENT VOWEL QOO TYPE TWO is thought to be a variant of U+17B1 ឺ 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 11-2*.

Figure 11-2. Common Multiple Forms in Khmer

ញញឹម *nyo + nyo + y + mo* [ɲɔ̃ɲum] “to smile”
 មីណើម *ca + i + nyo + coeng + ca + oe + mo* [ceŋcaəm] “eyebrow”
 ស្ងប់ *sa + coeng nyo + ba + bantoc* [sɔ̃ɔp] “to respect”
 កញ្ញា *ka + nyo + coeng + nyo + aa* [kaɲna:] “girl, Miss, September”
 ឲ្យ *qoo-2 + coeng + yo* (= ឲ្យ *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 QAA 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 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 KHMER VOWEL INHERENT AQ and U+17B5 KHMER VOWEL INHERENT AA 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 KHMER SIGN BEYYAL 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:

្ក្ក្ក្ក *khan + ba + e + khan*
 -្ក្ក- *en dash + ba + e + en dash*
 ្ក្ក ្ក្ក *khan + lo + khan*
 -្ក្ក- *en dash + lo + en dash*

Ordering of Syllable Components. The standard order of components in an orthographic syllable as expressed in BNF is

$$B \{R \mid C\} \{S \{R\}^* \{Z\} V\} \{O\} \{S\}$$

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 *khā* 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* [k^hnom] “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 11-3*.

Figure 11-3. Examples of Syllabic Order in Khmer

្រ to + e [tè:] “much”
 ្រ្រីន ca + coeng + ro + oe + no [craən] “much”
 ស្រ្រាម sa + ngo + coeng + ko + coeng + ro + aa + mo [sɔŋkrèəm] “war”
 ្រ្រី ha + oe + coeng + yo [haəi] “already”
 ស្រ្រ sa + nyo + coeng + nyo + aa [sajna:] “sign”
 ស្រ្រ sa + triisap + ii [si:] “eat”
 ្រ 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 11-3*.

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 ZERO WIDTH NON-JOINER. This provides Khmer with a fixed order of character placement, making it easier to search for words in a document.

្រ្រ mo + muusikatoan + coeng + ngo + ai [mɲai] “one day”
 ្រ្រត្រ mo + triisap + coeng + ha + ae + ta + lek too [mhè:tmhè:t]
 “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 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 ZERO WIDTH NON-JOINER is inserted before the vowel sign, as shown in the following examples:

្រ្រ ្រ ba + ^[ZW] + triisap + ii + yo + ae + ro [biyè:] “beer”
 ្រ្រ្រ្រ ba + coeng + ro + ta + yy + ngo + qa + ^[ZW] + triisap + y +
 reahmuk [prətə:ŋtuh] “urgent, too busy”
 ្រ្រ្រ្រ ba + coeng + ro + ta + yy + ngo + qa + triisap + y + reahmuk

Ligature Control. In the *askaa 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 ZERO WIDTH NON-JOINER may be inserted before the vowel sign to explicitly suppress such a ligature, as shown in *Figure 11-4* for the word “savant,” pronounced [vitu:].

Figure 11-4. Ligation in *Muul* Style in Khmer

វិទូ	vo + i + to + uu	(<i>askaa crieng</i> font)
វិទូ, វិទូ	vo + i + to + uu	(ligature dependent on the <i>muul</i> font)
វិទូ	vo + ^[ZW] + i + to + uu	(^[ZW] to prevent the ligature in a <i>muul</i> font)
វិទូ	vo + ^[ZW] + i + to + uu	(^[ZW] to request the ligature in a <i>muul</i> 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 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 16-2*.

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.

11.5 Tai Le

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, 𑜋𑜧𑜨𑜫 𑜇𑜨 [tai² la⁶] (in older orthography 𑜋𑜧𑜨 𑜇𑜨). 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 𑜋, 𑜧, 𑜨, 𑜫, 𑜇 or 𑜨, it is displayed to the right of the letter, as shown in *Table 11-10*.

Table 11-10. Tai Le Tone Marks

Syllable	New Orthography	Old Orthography
<i>ta</i>	𑜋	𑜋
<i>ta</i> ²	𑜋𑜨	𑜋𑜨
<i>ta</i> ³	𑜋𑜧	𑜋𑜧
<i>ta</i> ⁴	𑜋𑜫	𑜋𑜫

Table 11-10. Tai Le Tone Marks (Continued)

Syllable	New Orthography	Old Orthography
<i>ta</i> ⁵	တၢ	တံ
<i>ta</i> ⁶	တၢင	တံ
<i>ti</i>	တံ	တံ
<i>ti</i> ²	တံၤ	တံ
<i>ti</i> ³	တံၤ	တံ
<i>ti</i> ⁴	တံၤ	တံ
<i>ti</i> ⁵	တံၤ	တံ
<i>ti</i> ⁶	တံၤ	တံ

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, as shown in *Table 11-11*.

Table 11-11. Myanmar Digits

Myanmar-Style Glyphs	Tai Le-Style Glyphs
၀ ၁ ၂ ၃ ၄ ၅ ၆ ၇ ၈ ၉	၀ ၁ ၂ ၃ ၄ ၅ ၆ ၇ ၈ ၉
၀ ၁ ၂ ၃ ၄ ၅ ၆ ၇ ၈ ၉	၀ ၁ ၂ ၃ ၄ ၅ ၆ ၇ ၈ ၉

Punctuation. Both CJK punctuation and Western punctuation are used. Typographically, European digits are about the same height and depth as the tall characters [၂] and [၃]. In some fonts, the baseline for punctuation is the depth of those characters.

11.6 New Tai Lue

New Tai Lue: U+1980–U+19DF

The New Tai Lue script, also known as Xishuang Banna Dai, is used mainly in southern China. The script was developed in the twentieth 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 twentieth 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 11.7, 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. By contrast, Lanna uses both spacing vowel signs and nonspacing vowel signs, which appear above or below the consonants they modify.

Syllabic Structure. All vowel signs in New Tai Lue are considered combining characters and follow their base consonants in the text stream. Where a syllable is composed of a vowel sign to the left and a vowel or tone mark on the right of the consonant, a sequence of characters is used, in the order *consonant + vowel + tone mark*, as shown in Table 11-12.

Table 11-12. New Tai Lue Vowel Placement

ꨀ	ka +	ꨀ	e +	ꨀ	t1	→	ꨀꨀꨀ	[ke: ²]		
ꨀ	ka +	ꨀ	e +	ꨀ	i	→	ꨀꨀꨀ	[kə: ¹]		
ꨀ	ka +	ꨀ	e +	ꨀ	iy	→	ꨀꨀꨀ	[kəi ¹]		
ꨀ	ka +	ꨀ	e +	ꨀ	iy +	ꨀ	t1	→	ꨀꨀꨀꨀ	[kəi ²]
ꨀ	ka +	ꨀ	e +	ꨀ	iy +	ꨀ	t2	→	ꨀꨀꨀꨀ	[kəi ³]

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 11-13.

Table 11-13. New Tai Lue Registers and Tones

Display	Sequence	Register	Tone Mark	Tone	Transcription
ꨀ	ka ^h	high		1	[ka ¹]
ꨀꨀ	ka ^h + t1	high	t1	2	[ka ²]
ꨀꨀ	ka ^h + t2	high	t2	3	[ka ³]
ꨀ	ka ^l	low		4	[ka ⁴]
ꨀꨀ	ka ^l + t1	low	t1	5	[ka ⁵]
ꨀꨀ	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 ꨀ NEW TAI LUE THAM DIGIT ONE. The regular *hora* form for the digit, U+19D1 ꨀ NEW TAI LUE DIGIT ONE, has the exact same glyph shape as a common New Tai Lue vowel, U+19B3 ꨀ 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.

11.7 Tai Tham

Tai Tham: U+1A20–U+1AAF

The Tai Tham (or Lanna) script is used for three living languages: Northern Thai (that is, Kam Mu'ang), Tai Lue, and Khün. In addition, the script is also used for Lao Tham (or Old

Lao) and other dialect variants in Buddhist palm leaves and notebooks. The script is also known as the Tham or Yuan script. 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 670,000 speakers of Tai Lue. Of those, people born before 1950 may be literate in the Tai Tham script. Younger speakers are taught the New Tai Lue script, instead. (See *Section 11.6, New Tai Lue*.) The Tai Tham script continues to be taught in the Tai Lue monasteries. There are 120,000 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 HAM 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 KAAAN, 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.

Linebreaking. Opportunities for linebreaking are lexical, but a linebreak may not be inserted between a base letter and a combining diacritic. There is no line-breaking hyphenation.

11.8 Tai Viet

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. Vowel 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 about thirty years ago, 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 11-14.

Table 11-14. Tai Viet Symbols and Punctuation

Code	Glyph	Name	Meaning
AADB	ꨀ	<i>kon</i>	person
AADC	ꨁ	<i>nueng</i>	one
AADD	ꨂ	<i>sam</i>	signals repetition of the previous word
AADE	ꨃ	<i>ho hoi</i>	beginning of text (used in songs and poems)
AADF	ꨄ	<i>koi koi</i>	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.

11.9 Kayah Li

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, o, i, ô) are written as independent spacing letters. Five others (u, 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.

11.10 Cham

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].

Three medial clusters of two consonant signs in a row occur: <-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 11-15.

Table 11-15. Cham Syllabic Structure

Class	Examples	Encoding
consonant or independent vowel		[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*.

11.11 Philippine Scripts

Tagalog: U+1700–U+171F

Hanunóo: U+1720–U+173F

Buhid: U+1740–U+175F

Tagbanwa: U+1760–U+177F

The first of these four scripts—Tagalog—is no longer used, whereas the other three—Hanunóo, Buhid, and Tagbanwa—are living scripts of the Philippines. 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 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 written in the Latin script.

The three living scripts—Hanunóo, Buhid, and Tagbanwa—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.

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.

Independent Vowel Letters. Philippine scripts have null consonants, which are used to write syllables that start with a vowel.

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* in his 1620 catechism in the Ilocano language, but this innovation did not seem to find favor with native users, who seem to have considered the script adequate without it (they preferred 𑀓𑀔𑀕 *kakapi* to 𑀓𑀔𑀕𑀖 *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, 𑀓𑀔𑀕 𑀖𑀗𑀘 *si apu ba upada* is, with the *pamudpod*, rendered more accurately as 𑀓𑀔𑀕 𑀖𑀗𑀘𑀙𑀚𑀛𑀜𑀝 *si aypud bay upadan*; the Hanunóo pronunciation is *si aypod bay upadan*. The Tagalog *virama* and Hanunóo *pamudpod* cancel only the inherent *-a*. No conjunct consonants are employed in the Philippine scripts.

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 11-16*.

Table 11-16. Hanunóo and Buhid Vowel Sign Combinations

Hanunóo			Buhid		
x	x + \bar{o}	x + \bar{u}	x	x + \bar{o}	x + \bar{u}

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. Tagalog makes use of only the latter; Hanunóo, Buhid, and Tagbanwa make use of both marks.

11.12 Buginese

Buginese: U+1A00–U+1A1F

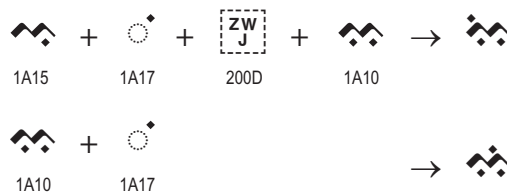
The Buginese script is used on the island of Sulawesi, mainly in the southwest. A variety of traditional literature has been printed in it. As of 1971, as many as 2.3 million speakers of Buginese were reported in the southern part of Sulawesi. The Buginese script is one of the easternmost of the Brahmi 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 lan-

guage with a rich traditional literature, is one of the foremost languages of Indonesia. The script was previously also used to write the Makassar, Bimanese, and Madurese languages.

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. Traditionally, a virama does not exist, but is included for modern usage in transcribing many non-Buginese words. This innovation is paralleled by a similar innovation in Hanunóo and Tagalog. The virama is always a visible sign. Because conjuncts are not formed in Buginese, U+200C ZERO WIDTH NON-JOINER is not necessary to force the display of the virama.

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 11-5*. 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 11-5*. The ligature for *îya* is not obligatory; it would be requested by inserting a *zero width joiner*.

Figure 11-5. 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+0662 ARABIC-INDIC DIGIT TWO or a doubling of the vowel sign (especially U+1A19 BUGINESE VOWEL SIGN E and U+1A1A BUGINESE VOWEL SIGN O) is used sometimes to denote word reduplication.

Numerals. There are no known digits specific to the Buginese script.

11.13 Balinese

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. A slightly modified version of the script is 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 or post-fixed, often with a change in shape. Table 11-17 shows the base consonants and their conjunct forms.

Table 11-17. Balinese Base Consonants and Conjunct Forms

Consonant	Base Form	Conjunct Form
ka	ꦏ	ꦏꦲ
kha	ꦏꦲꦲ	ꦏꦲꦲꦲ
ga	ꦒ	ꦒꦲ
gha	ꦒꦲꦲ	ꦒꦲꦲꦲ
nga	ꦒ	ꦒꦲ
ca	ꦕ	ꦕꦲ
cha	ꦕꦲꦲ	ꦕꦲꦲꦲ
ja	ꦗ	ꦗꦲ
jha	ꦗꦲꦲ	ꦗꦲꦲꦲ
nya	ꦒꦚ	ꦒꦚꦲ
tta	ꦠꦲ	ꦠꦲꦲ
ttha	ꦠꦲꦲꦲ	ꦠꦲꦲꦲꦲ
dda	ꦢ	ꦢꦲ
ddha	ꦢꦲꦲ	ꦢꦲꦲꦲ
nna	ꦒꦤ	ꦒꦤꦲ
ta	ꦠ	ꦠꦲ
tha	ꦠꦲꦲ	ꦠꦲꦲꦲ
da	ꦢ	ꦢꦲ
dha	ꦢꦲꦲ	ꦢꦲꦲꦲ
na	ꦒ	ꦒꦲ
pa	ꦥ	ꦥꦲ
pha	ꦥꦲꦲ	ꦥꦲꦲꦲ
ba	ꦁ	ꦁꦲ
bha	ꦁꦲꦲ	ꦁꦲꦲꦲ
ma	ꦩ	ꦩꦲ
ya	ꦪ	ꦪꦲ
ra	ꦫ	ꦫꦲ
la	ꦭ	ꦭꦲ
wa	ꦮ	ꦮꦲ

Table 11-17. Balinese Base Consonants and Conjunct Forms (Continued)

Consonant	Base Form	Conjunct Form
<i>ssa</i>	ꦱꦱ	ꦱꦱꦱ
<i>sha</i>	ꦱ	ꦱꦱ
<i>sa</i>	ꦱ	ꦱꦱꦱ
<i>ha</i>	ꦱ	ꦱꦱ
<i>r</i>	ꦱ	ꦱꦱ

The seven letters U+1B45 BALINESE LETTER KAF SASAK through U+1B4B BALINESE LETTER ASYURA SASAK are base consonant extensions for the Sasak language. Their base forms and conjunct forms are shown in *Table 11-18*.

Table 11-18. Sasak Extensions for Balinese

Consonant	Base Form	Conjunct Form
<i>kaf</i>	ꦱꦱ	ꦱꦱꦱ
<i>khot</i>	ꦱꦱꦱ	ꦱꦱꦱꦱ
<i>tzir</i>	ꦱꦱ	ꦱꦱꦱ
<i>ef</i>	ꦱ	ꦱꦱ
<i>ve</i>	ꦱ	ꦱꦱ
<i>zal</i>	ꦱꦱ	ꦱꦱꦱ
<i>asyura</i>	ꦱꦱꦱ	ꦱꦱꦱꦱ

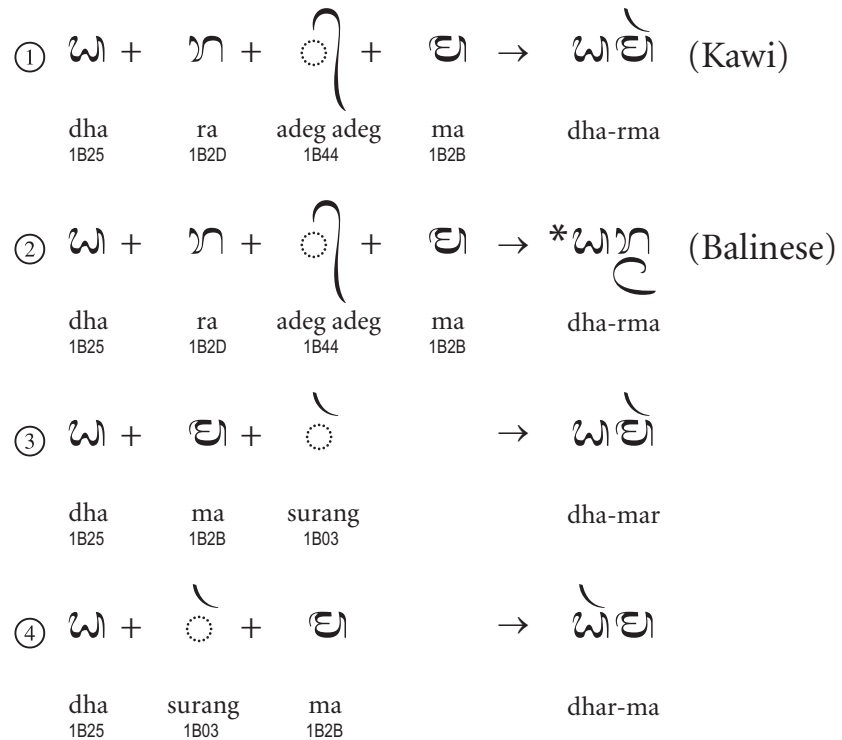
Balinese dependent vowel signs are used in a manner similar to that employed by other Brahmic scripts.

Independent vowels are used in a manner similar to that seen in other Brahmic scripts, with a few differences. For example, U+1B05 BALINESE LETTER AKARA and U+1B0B BALINESE LETTER RA REPA can be treated as consonants; that is, they can be followed by *adeg*. In Sasak, the vowel letter *akara* can be followed by an explicit *adeg adeg* ꦱꦱꦱꦱ in word- or syllable-final position, where it indicates the glottal stop; other consonants can also be subjoined to it.

Behavior of ra. Unlike most Brahmi-derived scripts, a Balinese *ra* that starts a sequence of consonants without intervening vowels is represented by U+1B03 BALINESE SIGN SURANG over the preceding syllable, as shown in the fourth example in *Figure 11-6*. The inherited Kawi form of the script used a *repha* glyph in the same way as many Brahmic scripts do. This is seen in the first example in *Figure 11-6*, where the sequence <*ra, virama, ma*> is rendered with the *repha* glyph. However, because many syllables end in *-r* in the Balinese language, this written form was historically reanalyzed, and is now pronounced *damar* in Balinese, as shown in the third example. In Balinese, the character sequence used in Kawi to spell *dharma* would render as shown in the second example, where the base letter *ra* with a subjoined *ma* is not well formed for the writing system.

Because of its relationship to *ra*, *surang* should be treated as equivalent to *ra* for searching and sorting purposes. Two other combining signs are also equivalent to base letters for

Figure 11-6. Writing *dharma* in Balinese



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 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* Pak Rërëh “Mr Rërëh”, the postfixed 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* Krësna .

Rendering. The vowel signs /u/ and /u:/ take different forms when combined with subscripted consonant clusters, as shown in *Table 11-19*. The upper limit of consonant clusters is three, the last of which can be *-ya*, *-wa*, or *-ra*.

Table 11-19. Balinese Consonant Clusters with u and u:



Syllable	Glyph
<i>kyu</i>	kyu
<i>kyú</i>	kyú
<i>kwu</i>	kwu
<i>kwú</i>	kwú
<i>kru</i>	kru

Table 11-19. Balinese Consonant Clusters with u and u: (Continued)

Syllable	Glyph
<i>krú</i>	
<i>kryu</i>	
<i>kryú</i>	
<i>skru</i>	
<i>skrú</i>	

Nukta. The combining mark U+1B34 BALINESE SIGN REREKAN (*nukta*) and a similar sign in Javanese are used to extend the character repertoire for foreign sounds. In recent times, Sasak users have abandoned the Javanese-influenced *rerekan* in favor of the series of modified letters shown in Table 11-18, also making use of some unused Kawi letters for these Arabic sounds.

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. Both U+1B5A BALINESE PANTI and U+1B5B BALINESE PAMADA are used to begin a section in text. U+1B5D BALINESE CARIK PAMUNGKAH is used as a colon. U+1B5E BALINESE CARIK SIKI and U+1B5F BALINESE CARIK PAREREN are used as comma and full stop, respectively. At the end of a section,  *pasalinan* and  *carik agung* may be used (depending on which sign began the section). They are encoded using the punctuation ring U+1B5C BALINESE WINDU together with *carik pareren* and *pamada*.

Hyphenation. Traditional Balinese texts are written on palm leaves; books of these bound leaves together are called *lontar*. U+1B60 BALINESE PAMENENG is inserted in *lontar* texts where a word must be broken at the end of a line (always after a full syllable). This sign is not used as a word-joining hyphen—it is used only in line breaking.

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 BALINESE LETTER UKARA and U+1B27 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 BALINESE MUSICAL SYMBOL DENG and U+1B68 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.

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.

11.14 Javanese

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 or postfixed, often with a change in shape.

Vocalic liquids (*r* and *l*) are treated as consonant letters in Javanese; they are not independent vowels with dependent vowel equivalents, as is the case in Balinese or Devanagari. Short and long versions of the *vocalic-l* are separately encoded, as U+A98A JAVANESE LETTER NGA LELET and U+A98B JAVANESE LETTER NGA LELELT RASWADI. In contrast, the long version of the *vocalic-r* is represented by a sequence of the short vowel U+A989 JAVANESE LETTER PA CERK followed by the dependent vowel sign *-aa*, U+A9B4 JAVANESE SIGN TARUNG, serving as a length mark in this case.

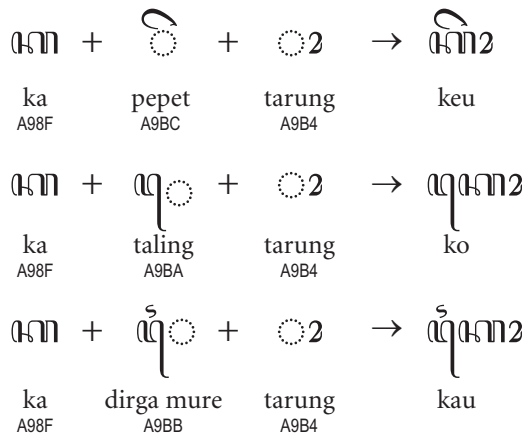
U+A983 JAVANESE SIGN CECAK TELU is a diacritic used with various consonantal base letters to represent foreign sounds. Typically these diacritic-marked consonants are used for sounds borrowed from Arabic.

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 SIGN TARUNG as a length mark.

Dependent Vowels. Javanese—unlike Balinese—represents multi-part 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 multi-part 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 11-7*.

Figure 11-7. Representation of Javanese Two-Part Vowels



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 two medial consonant signs, U+A9BE JAVANESE CONSONANT SIGN PENGKAL and U+A9BF JAVANESE CONSONANT SIGN CAKRA, which represent *-y-* and *-r-* respectively. These medial consonant signs contrast with the subjoined forms of the letters *ya* and *ra*. The subjoined forms may indicate a syllabic boundary, whereas *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+A9C8 JAVANESE PADA LINGSA is often used with digits in order to help to distinguish numbers from sequences of letters. When Javanese personal names are abbreviated, the letters are followed, not preceded, by PADA LINGSA.

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.

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 PADA 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.

Ordering of Syllable Components. The order of components in an orthographic syllable as expressed in BNF is:

$$\{C\} C \{\{R\}Y\} \{V\{A\}\} \{Z\}$$

where

C is a letter (consonant or independent vowel), or a consonant followed by the diacritic U+A9B3 JAVANESE SIGN CECAK TELU

F is the virama, U+A9C0 JAVANESE PANGKON

R is the medial -ra, U+A9BF JAVANESE CONSONANT SIGN CAKRA

Y is the medial -ya, U+A9BE JAVANESE CONSONANT SIGN PENGKAL

V is a dependent vowel sign

A is the dependent vowel sign -aa, U+A9B4 JAVANESE VOWEL SIGN TARUNG

Z is a consonant sign: U+A980, U+A981, U+A982, or U+A983

Linebreaking. Opportunities for linebreaking 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.

11.15 Rejang

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.

11.16 Batak

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. Batak uses a vowel killer which is called *pangolat* in Mandailing, Pakpak, and Toba. In Karo the killer is called *penengen*, and in Simalungun it is known as *panongonan*. The appearance of the killer differs between some of the dialects. Consonant conjuncts are not formed. 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.

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 disambiguate 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.

Linebreaking. Opportunities for a linebreak occur after any full orthographic syllable, defined as C(V(C_s|C_d)) where a consonant C may be followed by a vowel sign V which may be followed either by a consonant sign C_s (-ng or -h) or a killed final consonant C_d.

11.17 Sundanese

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 descendent 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 *Sunda 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). An explicit *virama*, U+1BAA SUNDANESE SIGN PAMAAEH, is used to indicate the absence, or “killing,” of the inherent vowel. Sundanese does not use the *virama* to cluster consonants or build consonant conjuncts.

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. The script also has independent vowel letters.

Three final consonants are separately encoded as combining marks: *-ng*, *-r*, and *-h*. These are analogues of Brahmic anusvara, repha, and visarga, respectively.

Consonant Additions. Two supplemental consonant letters have been added to the script recently: 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 is shown in more detail in Table 11-20.

Table 11-20. Sundanese Syllabic Structure

Class	Examples	Encoding
consonant or independent vowel	ᮘ	[U+1B83..U+1BA0, U+1BAE, U+1BAF]
consonant sign -ya, -ra, -la	ᮘᮓ, ᮘᮔ, ᮘᮕ	[U+1BA1..U+1BA3]
dependent vowel, virama	ᮘᮖ, ᮘᮗ	[U+1BA4..U+1BA9, U+1BAA]
final consonant	ᮘᮙ	[U+1B80..U+1B82]

The *virama* 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 PANAE LAENG 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.

Chapter 12

East Asian Scripts

This chapter presents the following scripts:

<i>Han</i>	<i>Hiragana</i>	<i>Hangul</i>
<i>Bopomofo</i>	<i>Katakana</i>	<i>Yi</i>

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 12.1, 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 12.4, 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 12.6, 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 twentieth 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 12.7, Yi*.

Bopomofo, discussed in Section 12.3, *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.

12.1 Han

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 ideo-

graphic 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.

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 two subsections, “CJK Standards” and “Blocks Containing Han Ideographs,” describe the character set standards used as sources and the way in which the Unicode Standard divides Han ideographs into blocks. These subsections are 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*.

CJK Standards

The Unicode Standard draws its unified Han character repertoire of 75,215 characters from a number of different character set standards. These standards are grouped into nine sources, as indicated in *Table 12-1*. The primary work of unifying and ordering the characters from these sources was done by the Ideographic Rapporteur Group (IRG), a subgroup of ISO/IEC JTC1/SC2/WG2.

Table 12-1. Sources for Unified Han

G source:	G0	GB 2312-80
	G1	GB 12345-90 with 58 Hong Kong and 92 Korean “Idu” characters
	G3	GB 7589-87 unsimplified forms
	G5	GB 7590-87 unsimplified forms
	G7	General Purpose Hanzi List for Modern Chinese Language, and General List of Simplified Hanzi
	GS	Singapore Characters
	G8	GB 8565-88
	G9	GB18030-2000
	GE	GB 16500-95
	G_4K	Siku Quanshu
	G_BK	Chinese Encyclopedia
	G_CH	Ci Hai
	G_CY	Ci Yuan
	G_CYY	Chinese Academy of Surveying and Mapping Ideographs
	G_FZ	Founder Press System
	G_GH	Gudai Hanyu Cidian
	G_GJZ	Commercial Press Ideographs
	G_HC	Hanyu Dacidian
	G_HZ	Hanyu Dazidian ideographs
	G_IDC	ID system of the Ministry of Public Security of China, 2009
	G_JZ	Commercial Press Ideographs
	G_KX	Kangxi Dictionary ideographs 9th edition (1958) including the addendum
	G_XC	Xiandai Hanyu Cidian
	G_ZFY	Hanyu Fangyan Dacidian
	G_ZH	ZhongHua ZiHai
	G_ZJW	Yinzhou Jinwen Jicheng Yinde
H source:	H	Hong Kong Supplementary Character Set – 2008
M source:	MAC	Macao Information System Character Set

Table 12-1. Sources for Unified Han (Continued)

T source:	T1	TCA-CNS 11643-1992 1st plane
	T2	TCA-CNS 11643-1992 2nd plane
	T3	TCA-CNS 11643-1992 3rd plane with some additional characters
	T4	TCA-CNS 11643-1992 4th plane
	T5	TCA-CNS 11643-1992 5th plane
	T6	TCA-CNS 11643-1992 6th plane
	T7	TCA-CNS 11643-1992 7th plane
	TB	TB TCA-CNS Ministry of Education, Hakka dialect, May 2007
	TC	TCA-CNS 11643-1992 12th plane
	TD	TCA-CNS 11643-1992 13th plane
	TE	TCA-CNS 11643-1992 14th plane
	TF	TCA-CNS 11643-1992 15th plane
J source:	J0	JIS X 0208-1990
	J1	JIS X 0212-1990
	J3	JIS X 0213:2000 level-3
	J3A	JIS X 0213:2004 level-3
	J4	JIS X 0213:2000 level-4
	JA	Unified Japanese IT Vendors Contemporary Ideographs, 1993
	JH	Hanyo-Denshi Program, 2002–2009
	JK	Japanese KOKUJI Collection
	J_ARIB	Association of Radio Industries and Businesses (ARIB) ARIB STD-B24 Version 5.1, March 14, 2007
K source:	K0	KS X 1001:2004 (formerly KS C 5601-1987)
	K1	KS X 1002:2001 (formerly KS C 5657-1991)
	K2	PKS C 5700-1 1994
	K3	PKS C 5700-2 1994
	K4	PKS 5700-3:1998
	K5	Korean IRG Hanja Character Set 5th Edition: 2001
KP source:	KP0	KPS 9566-97
	KP1	KPS 10721-2000 and KPS 10721:2003
V source:	V0	TCVN 5773:1993
	V1	TCVN 6056:1995
	V2	VHN 01:1998
	V3	VHN 02: 1998
	V4	Dictionary on Nom 2006, Dictionary on Nom of Tay ethnic 2006, Lookup Table for Nom in the South 1994
U source:	UTC	The Unicode Technical Report #45, U-source Ideographs, September 2010
	UCI	The Unicode Technical Report #45, U-source Ideographs, September 2010
		KS C 5601-1987 (duplicate ideographs)
		ANSI Z39.64-1989 (EACC)
		Big-5 (Taiwan)
		CCCII, level 1
		GB 12052-89 (Korean)
		JEF (Fujitsu)
		PRC Telegraph Code
		Taiwan Telegraph Code (CCDC)
		Xerox Chinese
		Han Character Shapes Permitted for Personal Names (Japan)
		IBM Selected Japanese and Korean Ideographs

The G, T, H, M, J, K, KP, 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 seven sources are the submissions from Taiwan, the Hong Kong SAR, the Macao SAR, Japan, South and North Korea, and Vietnam, respectively.

The U source represents character repertoires of three different types. First, it includes character submissions from the Unicode Technical Committee to the IRG. These were used

by the IRG in the preparation of Extensions C and D and have U source references prefixed with “UTC-”. Second, corrections to IRG data sometimes leave unified ideographs without any official IRG source. Such ideographs are included in the U source and given U-source references with the prefix “UCI-”. U source indices for references with a “UTC-” or a “UCI-” prefix are documented in Unicode Technical Report #45, “U-Source Ideographs.” Finally, the U source includes ideographs from character set standards that were not submitted to the IRG by any member body, but which were used by the Unicode Technical Committee during the preparation of the initial set of Unified CJK Ideographs included in the Unicode Standard, Version 1.0.1—the set known as the URO. These characters do not have any formal U source indices.

For each of the IRG sources, the second column in the table lists an abbreviated source name and the third column lists a descriptive source name. The only characters without abbreviated source names are U source characters added prior to Extension C. The abbreviated names are used in various data files published by the Unicode Consortium and ISO/IEC to identify the specific IRG sources.

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 is used, for example, with GB 2312-80 and GB 12345-90, which have many ideographs in common. Characters in GB 12345-90 that are duplicates of characters in GB 2312-80 are 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. These multiply encoded ideographs are not included in the K source but are included in the U source. 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 1001:1998).

Blocks Containing Han Ideographs

Han ideographic characters are found in seven main blocks of the Unicode Standard, as shown in *Table 12-2*.

Table 12-2. 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	Uncommon, some in current use
CJK Compatibility Ideographs	F900–FAFF	Duplicates, unifiable variants, corporate characters
CJK Compatibility Ideographs Supplement	2F800–2FA1F	Unifiable variants

Characters in the four 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.

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. It 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 12-3*.

Table 12-3. Small Extensions to the URO

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

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 submitted to the IRG during 1998–2002.

The CJK Unified Ideographs Extension C and D blocks contain rare, historic, or uncommon characters that are not unifiable with characters in any previously encoded CJK Unified Ideographs block. Some Extension D characters are in current use, particularly for Cantonese special use characters in Hong Kong. Extension C ideographs were submitted to the IRG during 2002–2006. Extension D ideographs were submitted to the IRG during 2006–2009.

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 four unified ideograph blocks are not closed repertoires. Each contains 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 itself. 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)”).

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. They are also called *mana* (true names, as opposed to *kana*, false or borrowed names).

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 12-4*).

Table 12-4. Common Han Characters

<i>Han Character</i>	<i>Chinese</i>	<i>Japanese</i>	<i>Korean</i>	<i>English Translation</i>
天	tiān	ten, ame	chen	heaven, sky
地	dì	chi, tsuchi	ci	earth, ground
人	rén	jin, hito	in	man, person
山	shān	san, yama	san	mountain
水	shuǐ	sui, mizu	swu	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 *tau* 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 12-8*.)

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 12-1*).

Figure 12-1. 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 *chuuugoku* in Japanese, for example, may refer to China or to a district in central west Honshuu (see *Figure 12-2*).

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.

Figure 12-2. Semantic Context for Han Characters

$$\begin{array}{rcl} \text{中} & + & \boxed{\text{玉}} \\ \text{middle} & & \text{country} \end{array} = \begin{array}{l} 1. \text{China} \\ 2. \text{Chuugoku district of Honshuu} \end{array}$$

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 臺, 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 说 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 丨 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).

Dialects and Early Forms of Chinese. Chinese is not a single language, but a complex of spoken forms that share a single written form. Although these spoken forms are referred to as dialects, they are actually mutually unintelligible and distinct languages. Virtually all modern written Chinese is Mandarin, the dominant language in both the PRC and Taiwan. Speakers of other Chinese languages learn to read and write Mandarin, although they pronounce it using the rules of their own language. (This would be like having Spanish children read and write only French, but pronouncing it as if it were Spanish.) The major non-Mandarin Chinese languages are Cantonese (spoken in the Hong Kong and Macao SARs, in many overseas Chinese communities, and in much of Guangdong province), Wu, Min, Hakka, Gan, and Xiang.

Prior to the twentieth century, the standard form of written Chinese was literary Chinese, a form derived from the classical Chinese written, but probably not spoken by Confucius in the sixth century BCE.

The ideographic repertoire of the Unicode Standard is sufficient for all but the most specialized texts of modern Chinese, literary Chinese, and classical Chinese. Preclassical Chinese, written using *seal forms* or *oracle bone forms*, has not been systematically incorporated into the Unicode Standard, because those very early, historic forms differed substantially from the classic and modern forms of Han characters. They require investigation and encoding as distinct historic scripts.

Among modern Chinese languages, Cantonese is occasionally found in printed materials; the others are almost never seen in printed form. There is less standardization for the ideographic repertoires of these languages, and no fully systematic effort has been undertaken to catalog the nonstandard ideographs they use. Because of efforts on the part of the government of the Hong Kong SAR, however, the current ideographic repertoire of the Unicode Standard should be adequate for many—but not all—written Cantonese texts.

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 12-8* 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.

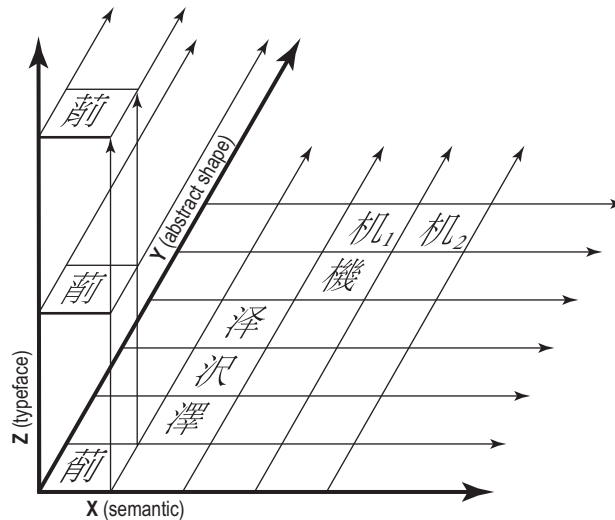
Principles of Han Unification

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, type-face form). These attributes are graphically represented in three dimensions according to the *X*, *Y*, and *Z* axes (see *Figure 12-3*).

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).

Figure 12-3. Three-Dimensional Conceptual Model



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 16.4, *Variation Selectors*.

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 12-4 illustrates six variants of the CJK ideograph meaning “sword.”

Figure 12-4. CJK Source Separation

劍 劍 劔 劔 劔 劔

“sword”

Each of the six variants in Figure 12-4 is separately encoded in one of the primary source standards—in this case, J0 (JIS X 0208-1990), as shown in Table 12-5.

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 pos-

Table 12-5. 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

sible 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 12-5, although visually quite similar, are nevertheless not unified because they are historically unrelated and have distinct meanings.

Figure 12-5. Not Cognates, Not Unified

 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.*

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 12-6 and Figure 12-7).

Ideograph Features. The following features of each ideograph to be compared are examined:

- Number of components
- Relative positions of components in each complete ideograph

Figure 12-6. Ideographic Component Structure

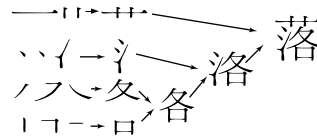
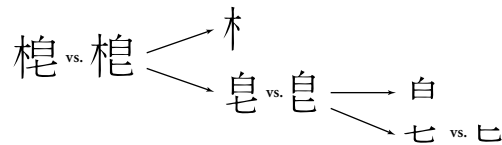


Figure 12-7. The Most Superior Node of an Ideographic Component



- 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 12-6* illustrate the reasons for not unifying characters, including typical differences in abstract character shape.

Table 12-6. Ideographs Not Unified

Characters	Reason
日 ≠ 曰	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
扞 ≠ 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 12-7*.

Table 12-7. Ideographs Unified

Characters	Reason
周 ≈ 周	Different writing sequence
雪 ≈ 雪	Differences in overshoot at the stroke termination
酉 ≈ 酉	Differences in contact of strokes
鉅 ≈ 鉅	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 ≈ 說	Difference in rotated strokes/dots ^a

- a. These ideographs (having the same abstract shape) would have been unified except for the Source Separation Rule.

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 12-8* with their priorities.

Table 12-8. Han Ideograph Arrangement

Priority	Dictionary	City	Publisher	Version
1	<i>KangXi Zidian</i>	Beijing	Zhonghua Bookstore, 1989	Seventh edition
2	<i>Dai Kan-Wa Jiten</i>	Tokyo	Taishuukan Shoten, 1986	Revised edition
3	<i>Hanyu Da Zidian</i>	Chengdu	Sichuan Cishu Publishing, 1986	First edition
4	<i>Dae Jaweon</i>	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 12-2*) 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 17.2, CJK Unified and Compatibility Ideographs*.

Radical-Stroke Indices

To expedite locating specific Han ideographic characters in the code charts, radical-stroke indices are provided on the Unicode web site. An interactive radical-stroke index page enables queries by specific radical numbers and stroke counts. Two fully formatted traditional radical-stroke indices are also posted in PDF format. The larger of those provides a radical-stroke index for all of the Han ideographic characters in the Unicode Standard, including CJK compatibility ideographs. There is also a more compact radical-stroke index limited to the IICore set of 9,810 CJK unified ideographs in common usage. The following text describes how radical-stroke indices work for Han ideographic characters and explains the particular adaptations which have been made for the Unicode radical-stroke indices.

Under the traditional radical-stroke system, each Han ideograph is considered to be written with one of a number of different character elements or radicals and a number of additional strokes. For example, the character 說 has the radical 言 and seven additional strokes. To find the character 說 within a dictionary, one would first locate the section for its radical, 言, and then find the subsection for characters with seven additional strokes.

This method is complicated by the fact that there are occasional ambiguities in the counting of strokes. Even worse, some characters are considered by different authorities to be written with different radicals; there is not, in fact, universal agreement about which set of radicals to use for certain characters, particularly with the increased use of simplified characters.

The most influential authority for radical-stroke information is the eighteenth-century *KangXi* dictionary, which contains 214 radicals. The main problem in using *KangXi* radicals today is that many simplified characters are difficult to classify under any of the 214 *KangXi* radicals. As a result, various modern radical sets have been introduced. None, however, is in general use, and the 214 *KangXi* radicals remain the best known. See “CJK and KangXi Radicals” in the following text.

The Unicode radical-stroke charts are based on the *KangXi* radicals. The Unicode Standard follows a number of different sources for radical-stroke classification. Where two sources are at odds as to radical or stroke count for a given character, the character is shown in *both* positions in the radical-stroke charts.

Simplified characters are, as a rule, considered to have the same radical as their traditional forms and are found under the appropriate radical. For example, the character 倪 is found under the same radical, 人, as its traditional form (倪).

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 pub-

lished 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.

CJK Unified Ideographs Extension B: U+2000–U+2A6D6

The ideographs in the CJK Unified Ideographs Extension B block represent an additional set of 42,711 ideographs beyond the 27,496 included in *The Unicode Standard, Version 3.0*. The same principles underlying the selection, organization, and unification of Han ideographs apply to these ideographs.

As with other Han ideograph blocks, the ideographs in the CJK Unified Ideographs Extension B block are derived from versions of national standards submitted to the IRG by its members. They may in some instances differ slightly from published versions of these standards.

CJK Unified Ideographs Extension C: U+2A700–U+2B734

The ideographs in the CJK Unified Ideographs Extension C block represent an additional 4,908 ideographs beyond the 70,229 included in *The Unicode Standard, Version 5.0*. The same principles underlying the selection, organization, and unification of Han ideographs apply to these ideographs.

CJK Unified Ideographs Extension D: U+2B740–U+2B81D

The ideographs in the CJK Unified Ideographs Extension D block represent an additional 222 ideographs beyond the 74,394 included in *The Unicode Standard, Version 5.2*. The same principles underlying the selection, organization, and unification of Han ideographs apply to these ideographs.

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 the CJK Unified Ideographs

Areas. 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 IDEOGRAPH-F900.

CJK Compatibility Supplement: U+2F80–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.

Kanbun: U+3190–U+319F

This block contains a set of Kanbun marks used in Japanese texts to indicate the Japanese reading order of classical Chinese texts. These marks are not encoded in any other current character encoding standards but are widely used in literature. They are typically written in an annotation style to the left of each line of vertically rendered Chinese text. For more details, see JIS X 4051.

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” in *Section 15.10, Enclosed and Square*.

CJK and KangXi Radicals: U+2E80–U+2FD5

East Asian ideographic *radicals* are ideographs or fragments of ideographs used to index dictionaries and word lists, and as the basis for creating new ideographs. The term *radical* comes from the Latin *radix*, meaning “root,” and refers to the part of the character under which the character is classified in dictionaries. See “Radical-Stroke Indices” earlier in this section for a more detailed discussion of how ideographic radicals are used in indices.

There is no single radical set in general use throughout East Asia. However, the set of 214 radicals used in the eighteenth-century *KangXi* dictionary is universally recognized.

The visual appearance of radicals is often very different when they are used as radicals than their appearance when they are stand-alone ideographs. Indeed, many radicals have multi-

ple graphic forms when used as parts of characters. A standard example is the water radical, which is written 水 when an ideograph and generally 氵 when part of an ideograph.

The Unicode Standard includes two blocks of encoded radicals: the KangXi Radicals block (U+2F00..U+2FD5), which contains the base forms for the 214 radicals, and the CJK Radicals Supplement block (U+2E80..U+2EF3), which contains a set of variant shapes taken by the radicals either when they occur as parts of characters or when they are used for simplified Chinese. These variant shapes are commonly found as independent and distinct characters in dictionary indices—such as for the radical-stroke charts in the Unicode Standard. As such, they have not been subject to the usual unification rules used for other characters in the standard.

Most of the characters in the CJK and KangXi Radicals blocks are equivalents of characters in the CJK Unified Ideographs block of the Unicode Standard. Radicals that have one graphic form as an ideograph and another as part of an ideograph are generally encoded in both forms in the CJK Unified Ideographs block (such as U+6C34 and U+6C35 for the water radical).

Standards. CNS 11643-1992 includes a block of radicals separate from its ideograph block. This block includes 212 of the 214 KangXi radicals. These characters are included in the KangXi Radicals block.

Those radicals that are ideographs in their own right have a definite meaning and are usually referred to by that meaning. Accordingly, most of the characters in the KangXi Radicals block have been assigned names reflecting their meaning. The other radicals have been given names based on their shape.

Semantics. Characters in the CJK and KangXi Radicals blocks should never be used as ideographs. They have different properties and meanings. U+2F00 KANGXI RADICAL ONE is not equivalent to U+4E00 CJK UNIFIED IDEOGRAPH-4E00, for example. The former is to be treated as a symbol, the latter as a word or part of a word.

The characters in the CJK and KangXi Radicals blocks are compatibility characters. Except in cases where it is necessary to make a semantic distinction between a Chinese character in its role as a radical and the same Chinese character in its role as an ideograph, the characters from the Unified Ideographs blocks should be used instead of the compatibility radicals. To emphasize this difference, radicals may be given a distinct font style from their ideographic counterparts.

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-ideographic 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.

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.

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*.

12.2 Ideographic Description Characters

Ideographic Description: U+2FF0–U+2FFB

Although the Unicode Standard includes more than 75,000 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 12 characters in the Ideographic Description block 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 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 Ideo-

graphic 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 block are 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. These historic scripts are not yet encoded in Version 6.1 of the Unicode Standard, but it is quite likely that one or more of them will be encoded eventually.

The general usefulness of Ideographic Description Sequences for describing unencoded characters and the applicability of the characters in the Ideographic Description 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.

```
IDS := Ideographic | Radical | Private Use
      | IDS_BinaryOperator IDS IDS
      | IDS_TertiaryOperator IDS IDS IDS
```

```
IDS_BinaryOperator := U+2FF0 | U+2FF1 | U+2FF4 | U+2FF5 | U+2FF6 | U+2FF7 |
                    U+2FF8 | U+2FF9 | U+2FFA | U+2FFB
```

```
IDS_TertiaryOperator := U+2FF2 | U+2FF3
```

Previous versions of the Unicode standard imposed various limits on the length of a sequence or parts of it. Those limits are no longer imposed by the standard.

The operators indicate the relative graphic positions of the operands running from left to right and from top to bottom. 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 12-8 illustrates the use of this grammar to provide descriptions of unencoded ideographs. Examples 9–13 illustrate more complex Ideographic Description Sequences showing the use of some of the less common operators.

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 1–3 in *Figure 12-8*) or because duplications appear within the Unicode Standard (examples 5 and 6 in *Figure 12-8*).

The Unicode Standard does not define equivalence for two Ideographic Description Sequences that are not identical. *Figure 12-8* contains numerous examples illustrating how different Ideographic Description Sequences might be used to describe the same ideograph.

Figure 12-8. Using the Ideographic Description Characters



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. As with ideographs proper, the Ideographic Variation Mark (U+303E) may be placed before an Ideographic Description Sequence to indicate that the description is merely an approximation of the original ideograph desired. A sequence of characters that includes an Ideographic Variation Mark is not an Ideographic Description 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. The Ideographic Description characters are found in GBK—an extension to GB 2312-80 that adds all Unicode ideographs not already in GB 2312-80. GBK is defined as a normative annex of GB 13000.1-93.

12.3 Bopomofo

Bopomofo: U+3100–U+312F

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 12-9*.

Table 12-9. 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. The character U+3127 BOPOMOFO LETTER I may be rendered as either a horizontal stroke or a vertical stroke. Often the glyph is chosen to stand perpendicular to the text baseline (for example, a horizontal stroke in vertically set text), but other usage is also common. In the Unicode Standard, the form shown in the charts is a vertical stroke; the horizontal stroke form is considered to be a rendering variant. The variant glyph is not assigned a separate character code.

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 refer-

ence 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. There are no standard Bopomofo letters for the phonetics of Cantonese or several other Southern Chinese dialects.

The small characters encoded at U+31B4..U+31B7 represent syllable-final consonants not present in standard Mandarin or in Mandarin dialects. They have the same shapes as Bopomofo “b”, “d”, “k”, and “h”, 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.

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.

Extended Bopomofo Tone Marks. In addition to the Mandarin tone marks enumerated in Table 12-9, 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 12-10. 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 12-10. 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.

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 Latin digits 0–9 to express changes in actual tonetic values resulting from juxtaposition of basic tones.

12.4 Hiragana and Katakana

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.

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.

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.

Kana Supplement U+1B000–U+1B0FF

The Kana Supplement block is intended for the encoding of historic and variant forms of Japanese kana characters, including those variants collectively known as *hentaigana* in Japanese.

Currently this block contains two kana which are of historical use only. These have not been used in Japanese orthography for about one thousand years. The character U+1B000 KATAKANA LETTER ARCHAIC E is an obsolete form of the letter U+30A8 KATAKANA LETTER E. In its pre-10th century use, this letter represented the sound “e”, and U+30A8 KATAKANA LETTER E represented the sound “ye”. The character U+1B001 HIRAGANA LETTER ARCHAIC YE represents a long-obsolete syllable that would have come between U+3086 HIRAGANA LETTER YU and U+3088 HIRAGANA LETTER YO. This sound merged with “e”, which is now represented by U+3048 HIRAGANA LETTER E. These relationships are illustrated in Figure 12-9.

Figure 12-9. Japanese Historic Kana for *e* and *ye*

Pronunciation:	<i>e</i>	<i>ye</i>
Kanji Source:	衣	江
Hiragana:	え	
Katakana:		エ

12.5 Halfwidth and Fullwidth Forms

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 IDEOGRAPHIC SPACE.

12.6 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 ancient syllable blocks used in Old Korean. 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*.

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 15.10, Enclosed and Square*, as well as *Section 12.5, Halfwidth and Fullwidth Forms*.

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.

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.

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.

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 half-width 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 12-11 illustrates how two Hangul compatibility jamo can be separated in display, even after transforming them with NFKD or NFKC.

Table 12-11. 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

Hangul Syllables: U+AC00–U+D7A3

The Hangul script used in the Korean writing system consists of individual consonant and vowel letters (jamo) that are visually combined into square display cells to form entire syllable blocks. Hangul syllables may be encoded directly as precomposed 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 thousands 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 12-12*.

Table 12-12. 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 block. Because the order of the syllables in the Hangul Syllables block reflects the preferred ordering, sequences of Hangul syllables for modern Korean may be collated with a simple binary comparison.

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.

12.7 Yi

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 twentieth 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 diacritic mark shaped like an inverted breve. For example, U+A001 YI SYLLABLE IX is the same as U+A002 YI SYLLABLE I plus a diacritic 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 diacritic 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 12.1, 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.

Chapter 13

Additional Modern Scripts

This chapter contains a collection of additional scripts in modern use that do not fit well into the script categories featured in other chapters:

<i>Ethiopic</i>	<i>Vai</i>	<i>Deseret</i>
<i>Mongolian</i>	<i>Bamum</i>	<i>Shavian</i>
<i>Osmanya</i>	<i>Cherokee</i>	<i>Lisu</i>
<i>Tifinagh</i>	<i>Canadian Aboriginal Syllabics</i>	<i>Miao</i>
<i>N’Ko</i>		

Ethiopic, Mongolian, 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, including shorthand. 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.

The Cherokee script is a syllabary developed between 1815 and 1821, to write the Cherokee language, still spoken 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.

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. Shavian is another phonemic alphabet, invented in the 1950s to write English. It was used to publish one book in 1962, but remains of some current interest.

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 shorthand 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.

13.1 Ethiopic

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 *Table 13-1*. 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 13-1. Labialized Forms in Ethiopic -WAA

-WAA Form	Encoded as	Not Used	Contrast
QWAA	U+124B ቧ	1247	U+1247 ቧ QOA
QHWAA	U+125B ባ	1257	
XWAA	U+128B ባ	1287	U+1287 ባ XOA
KWAA	U+12B3 ኣ	12AF	U+12AF ኣ KOA
KXWAA	U+12C3 ኣ	12BF	
GWAA	U+1313 ኣ	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 13-2*.

Table 13-2. Labialized Forms in Ethiopic -WE

“-WE” Form	Encoded as	Not Used
QWE	U+124D ቁላ	1249
QHWE	U+125D ቁላላ	1259
XWE	U+128D ጁላ	1289
KWE	U+12B5 ከላ	12B1
KXWE	U+12C5 ከላላ	12C1
GWE	U+1315 ጉላ	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.

Word Separators. The traditional word separator is U+1361 ETHIOPIC WORDSPACE (:). In modern usage, a plain white whitespace (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.

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 2345 is represented by

$$\begin{aligned} 2345 &= (20 + 3) * 100^1 + (40 + 5) * 100^0 \\ &= 20 \quad 3 \quad 100 \quad 40 \quad 5 \\ &= \text{TWENTY THREE HUNDRED FORTY FIVE} \\ &= 1373 \quad 136B \quad 137B \quad 1375 \quad 136D \quad \text{ጥፋፋፋፋፋፋ} \end{aligned}$$

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:

$$\begin{aligned} 2,345 &= \text{TWO thousand THREE HUNDRED FORTY FIVE} \\ &= 136A \quad 123A \quad 136B \quad 137B \quad 1375 \quad 136D \quad \text{ጥሺ.ፋፋፋፋፋፋ} \end{aligned}$$

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 three blocks of extensions to the Ethiopic script: Ethiopic Supplement U+1380..U+139F, Ethiopic Extended U+2D80..U+2DDE, and Ethiopic Extended-A U+AB00..U+AB2F. Those extensions cover such languages as Me’em, Blin, and Sebatbeit, 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 diacritic marks used for Basketo.

The Ethiopic Supplement block also contains a set of tonal marks. They are used in multi-line scored layout. Like other musical (an)notational systems of this type, these tonal marks require a higher-level protocol to enable proper rendering.

13.2 Mongolian

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 Mongolia proper (formally the Mongolian People’s Republic, 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.

The Mongolian block unifies Mongolian and the three derivative scripts Todo, Manchu, and Sibe. Each of the three derivative scripts shares some common letters with Mongolian,

and these letters are encoded only once. Each derivative script also has a number of modified letter forms or new letters, which are encoded separately.

Mongolian, Todo, and Manchu also have a number of special “Ali Gali” letters that are used for transcribing Tibetan and Sanskrit in Buddhist texts.

History. The Mongolian script was derived from the Uighur script around the beginning of the thirteenth century, during the reign of Genghis Khan. The Uighur script, which was in use from about the eighth to the fifteenth centuries, was derived from Sogdian Aramaic, a Semitic script written horizontally from right to left. Probably under the influence of the Chinese script, the Uighur script became rotated 90 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 Uighur script.

The Mongolian script has remained in continuous use for writing Mongolian within the Inner Mongolia Autonomous Region of the People’s Republic of China and elsewhere in China. However, in the Mongolian People’s Republic (present-day 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 became 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 Sibö, 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 90 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, with the glyphs rotated 90 degrees counterclockwise with respect to their orientation in the code charts. 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 13-1*. There are many such examples in Mongolian, but not in Todo, Manchu, or Sibe, which have largely eliminated ambiguous letters.

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 13-2*, where the letter *ba* combines with the letter *u* to form a ligature in the Mongolian word *abu* “father.”

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

Figure 13-1. Mongolian Glyph Convergence

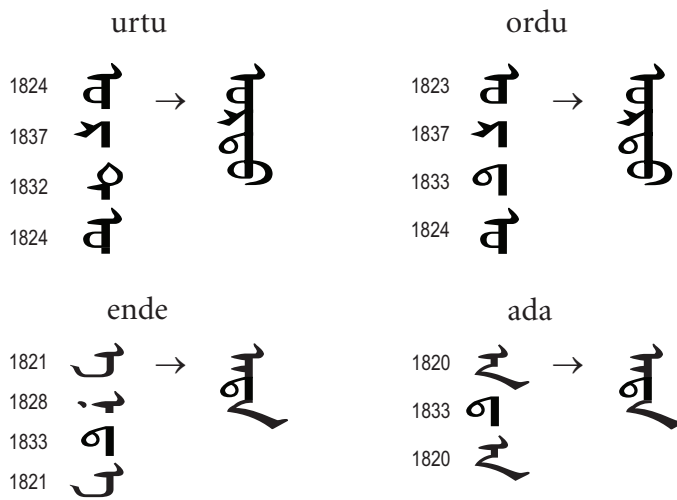
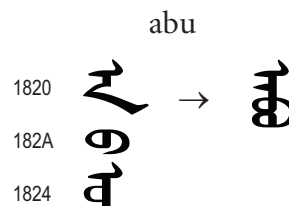
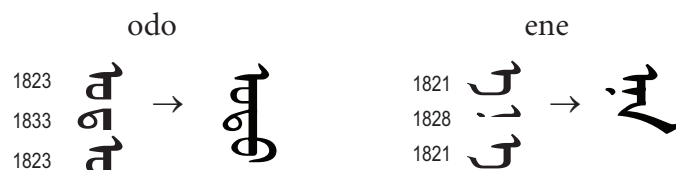


Figure 13-2. Mongolian Consonant Ligation



often the same as the initial form, but the final form is always distinct from the initial or medial form. *Figure 13-3* 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 13-3. 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 16, 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 isolate form.
- <1820 200D> selects the initial form.
- <200D 1820> selects the final form.
- <200D 1820 200D> selects the 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 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)

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 13-4 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 13-4. Mongolian Free Variation Selector



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 16.4, Variation Selectors*. Older implementations of Mongolian free variation selectors may therefore interpret a sequence such as a base character followed by 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 MONGOLIAN 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.

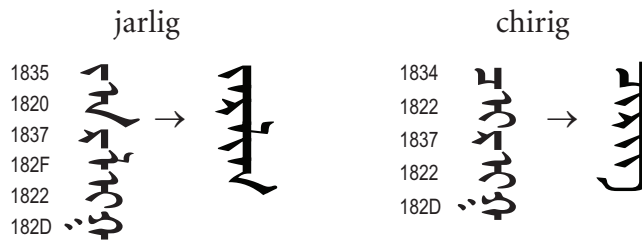
Vowel harmony is an important element of the encoding model, as the gender of a word determines the glyph form of the velar series of consonant letters for Mongolian, Todo, Sibe, and Manchu. In each script, the velar letters have both masculine and feminine forms. For Mongolian and Todo, the masculine 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 13-5*, 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 *chirig* “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.

Narrow No-Break Space. In Mongolian, Todo, Manchu, and Sibe, certain grammatical suffixes are separated from the stem of a word or from other suffixes by a narrow gap. There are many such suffixes in Mongolian, usually occurring in masculine and feminine pairs (for example, the dative suffixes *-dur* and *-dür*), and a stem may take multiple suffixes. In contrast, there are only six separated suffixes for Manchu and Sibe, and stems do not take more than one suffix at a time.

Figure 13-5. Mongolian Gender Forms

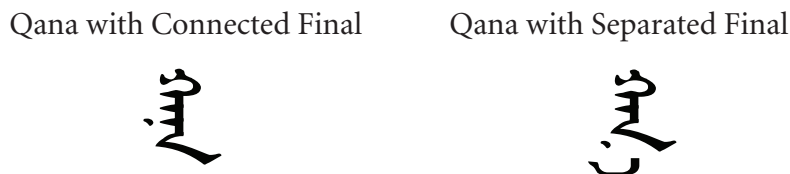


As any suffixes are considered to be an integral part of the word as a whole, a line break opportunity does not occur before a suffix, and the whitespace is represented using U+202F NARROW NO-BREAK SPACE (NNBSP). For a Mongolian font it is recommended that the width of NNBSP should be one-third the width of an ordinary space (U+0020 SPACE).

NNBSP affects the form of the preceding and following letters. The final letter of the stem or suffix preceding the NNBSP takes the final positional form, whereas the first letter of the suffix following NNBSP may take the normal initial form, a variant initial form, a medial form, or a final form, depending on the particular suffix.

Mongolian Vowel Separator. In Mongolian, the letters *a* (U+1820) and *e* (U+1821) in a word-final position may take a “forward tail” form or a “backward tail” form depending on the preceding consonant that they are attached to. In some words, a final letter *a* or *e* is separated from the preceding consonant by a narrow gap, in which case the vowel always takes the “forward tail” form. U+180E MONGOLIAN VOWEL SEPARATOR (MVS) is used to represent the whitespace that separates a final letter *a* or *e* from the rest of the word. MVS is very similar in function to NNBSP, as it divides a word with a narrow non-breaking whitespace. Whereas NNBSP marks off a grammatical suffix, however, the *a* or *e* following 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. For example, the word *qana* <182C, 1820, 1828, 1820> without a gap before the final letter *a* means “the outer casing of a vein,” whereas the word *qana* <182C, 1820, 1828, 180E, 1820> with a gap before the final letter *a* means “the wall of a tent,” as shown in *Figure 13-6*.

Figure 13-6. Mongolian Vowel Separator



The MVS has a twofold effect on shaping. On the one hand, it always selects the forward 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.

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 Todo, U+1806 MONGOLIAN TODO SOFT HYPHEN is used at the beginning of the second line to indicate resumption of a broken word. It functions 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, which is derived from the medial form of the letter *a* (U+1820), 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.

13.3 Osmanya

Osmanya: U+10480–U+104AF

The Osmanya script, which in Somali is called **ሒሳብ ጽሑፍ** *far Soomaali* “Somali writing” or **ሒሳብ ጽሑፍ** *Cismaanya*, was devised in 1920–1922 by **ሒሳብ ጽሑፍ ጽሑፍ ስራዎች** (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 **ǧ** OSMANYA LETTER AA, U+1049C **ⵓ** OSMANYA LETTER EE, and U+1049D **ⵓ** OSMANYA LETTER OO; long *uu* and *ii* are written with the consonants U+10493 **ⵓ** OSMANYA LETTER WAW and U+10495 **ⵓ** 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).

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.

13.4 Tifinagh

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.

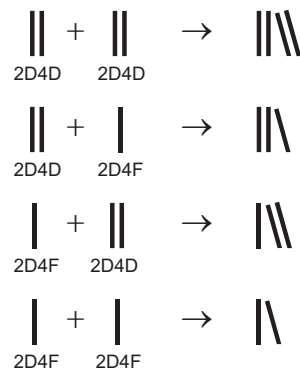
Directionality. Historically, Berber texts did not have a fixed direction. Early inscriptions were written horizontally from left to right, from right to left, vertically (bottom to top, top to bottom); boustrophedon directionality was also known. Modern-day Berber script is most frequently written in horizontal lines from left to right; therefore 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” /i:/, and <U+2D31, U+0302> represents a “p”. Some long vowels are represented using two diacritical marks above. A long “e” /e:/ 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.

Contextual Shaping. Contextual shaping of some consonants occurs when U+2D4D TIFINAGH LETTER YAL or U+2D4F TIFINAGH LETTER YAN are doubled or when both characters appear together in various sequences. The shaping distinguishes the characters when they appear next to each other. In contextual shaping, the second character is shifted vertically, or it can be slanted. *Figure 13-7* illustrates the use of contextual shaping.

Figure 13-7. Tifinagh Contextual Shaping



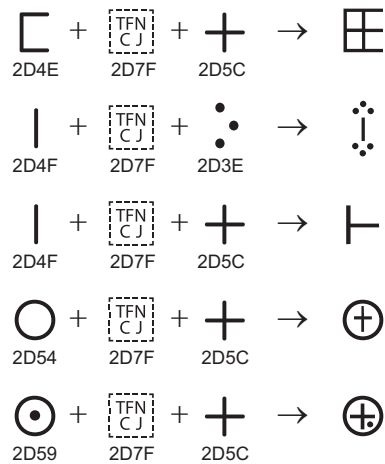
Bi-Consonants. Bi-consonants are additional letterforms used in the Tifinagh script, particularly for Tuareg, to represent a consonant cluster—a sequence of two consonants without an intervening vowel. These bi-consonants, sometimes also referred to as bigraphs, are not directly encoded as single characters in the Unicode Standard. Instead, they are represented as a sequence of the two consonant letters, separated either by U+200D ZERO WIDTH JOINER or by U+2D7F TIFINAGH CONSONANT JOINER.

When a bi-consonant is considered obligatory in text, it is represented by the two consonant letters, with U+2D7F TIFINAGH CONSONANT JOINER inserted between them. This use of U+2D7F is comparable in function to the use of U+0652 ARABIC SUKUN 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 TIFINAGH CONSONANT JOINER 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 13-8*.

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 TIFINAGH CONSONANT JOINER 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 ZERO WIDTH JOINER inserted between them, as a hint that the ligated form is preferred. If a ren-

Figure 13-8. Tifinagh Consonant Joiner and Bi-consonants



dering 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.

13.5 N’Ko

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 $\hat{\text{e}}\text{b}$, although each may pronounce it differently.

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 WOLOS0 and U+07E7 NKO LETTER NYA WOLOS0, 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, **ɓɓ** <ba, la, oo> is pronounced [bolo], but in a foreign syllable to be pronounced [blo], the *dagbasinna* is inserted for **ɓɓɓ** <ba, dagbasinna, la, oo> to show that a consonant cluster is intended.

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 13-9*. 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.

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.

Table 13-3 shows the use of the tone diacritics when applied to vowels.

Table 13-3. 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.

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 13-4*.

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 13-9*.

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

Table 13-4. Other 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	[b ^h]
	TA	[t] or Arabic ط TAH
	JA	[z] or Arabic ز ZAIN
	CA	[ð] or Arabic ذ THAL and also French [ʒ]
	DA	[d̥] or Arabic ض ZAD
	RA	French [ʀ]
	SA	[ʃ] or Arabic ش SHEEN
	GBA	[g]
	FA	[v]
	KA	[ħ] or Arabic ح KHAH
	LA	[l ^h]
	MA	[m ^h]
	NYA	[n ^h]
HA	[h] or Arabic ه HAH	
YA	[y ^h]	
U+07F3 NKO COMBINING DOUBLE DOT ABOVE	A	[ʕa] or Arabic ع AIN + A
	EE	French [ə]
	U	French [y]
	JA	[z] or Arabic ظ ZAH
	DA	[d ^h]
	SA	[θ] or Arabic ث THEH
	GBA	[kp]

Figure 13-9. Examples of N’Ko Ordinals

ḥ	1st
ḥ	2nd
ḥ	3rd
ḥḥḥ	144th

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 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.

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”.

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 13-5.

Table 13-5. N’Ko Letter Shaping

Character	X _n	X _r	X _m	X _l
A	Ɑ	Ɱ	Ɐ	Ɒ
EE	ⱱ	Ⱳ	ⱳ	ⱴ
I	Ⱶ	ⱶ	ⱷ	ⱸ
E	ⱹ	ⱺ	ⱻ	ⱼ
U	ⱽ	Ȿ	Ɀ	Ⳁ
OO	ⳁ	Ⳃ	ⳃ	Ⳅ
O	ⳅ	Ⳇ	ⳇ	Ⳉ
DAGBASINNA	ⳉ	Ⳋ	ⳋ	Ⳍ
N	ⳍ	Ⳏ	ⳏ	Ⳑ
BA	ⳑ	Ⳓ	ⳓ	Ⳕ
PA	ⳕ	Ⳗ	ⳗ	Ⳙ
TA	ⳙ	Ⳛ	ⳛ	Ⳝ
JA	ⳝ	Ⳟ	ⳟ	Ⳡ
CHA	ⳡ	Ⳣ	ⳣ	ⳤ
DA	⳥	⳦	⳧	⳨
RA	⳩	⳪	Ⳬ	ⳬ
RRA	Ⳮ	ⳮ	⳯	⳰
SA	⳱	Ⳳ	ⳳ	⳴
GBA	⳵	⳶	⳷	⳸
FA	⳹	⳺	⳻	⳼
KA	⳽	⳾	⳿	ⴀ
LA	ⴁ	ⴂ	ⴃ	ⴄ
NA WOLOSO	ⴅ	ⴆ	ⴇ	ⴈ
MA	ⴉ	ⴊ	ⴋ	ⴌ
NYA	ⴍ	ⴎ	ⴏ	ⴐ

Table 13-5. N’Ko Letter Shaping (Continued)

Character	X _n	X _r	X _m	X _l
NA	᠋	᠋	᠋	᠋
HA	᠋	᠋	᠋	᠋
WA	᠋	᠋	᠋	᠋
YA	᠋	᠋	᠋	᠋
NYA WOLOSŌ	᠋	᠋	᠋	᠋
JONA JA	᠋	᠋	᠋	᠋
JONA CHA	᠋	᠋	᠋	᠋
JONA RA	᠋	᠋	᠋	᠋

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 NKO LAJANYALAN is sometimes used like U+0640 ARABIC TATWEEL to justify lines, although Latin-style justification where space is increased tends to be more common.

13.6 Vai

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 Mòṃṃṃ Duwalu Bukelē of Jondū, 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.

13.7 Bamum

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 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.

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.

13.8 Cherokee

Cherokee: U+13A0–U+13FF

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 **ᎠᎯᏍᎦ** *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. The first font for Cherokee was designed by Dr. Samuel A. Worcester. Using fonts available to him, he assigned a number of Latin letters to the Cherokee syllables. At this time the Cherokee letter “HV” was dropped, and the Cherokee syllabary reached its current size of 85 letters. Dr. Worcester’s press printed 13,980,000 pages of Native American-language text, most of it in Cherokee.

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.

Case and Spelling. The Cherokee script is caseless, although for purposes of emphasis occasionally one letter will be made larger than the others. Cherokee spelling is not standardized: each person spells as the word sounds to him or her.

Numbers. Although Sequoyah invented a Cherokee number system, it was not adopted and is not encoded here. 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.

Rendering and Input. Cherokee is a left-to-right script, which requires no combining characters. 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.

Punctuation. Cherokee uses standard Latin punctuation.

Standards. There are no other encoding standards for Cherokee.

13.9 Canadian Aboriginal Syllabics

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:

V	^	>	<	<
PE	PI	PO	PA	P

Some variations in vowels also occur. For example, in Inuktitut usage, the syllable U+1450 \supset 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 \supset 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 \wedge 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.

Extensions. A few additional syllables in the range U+166E..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.

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).

13.10 Deseret

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 13-10*.

Figure 13-10. Short Words Equivalent to Deseret Letter Names

ᄁ	AY is written for <i>eye</i> or <i>I</i>
ᄂ	YEE is written for <i>ye</i>
ᄃ	BEE is written for <i>be</i> or <i>bee</i>
ᄄ	GAY is written for <i>gay</i>
ᄅ	THEE is written for <i>the</i> or <i>thee</i>

Phonetics. An approximate IPA transcription of the sounds represented by the Deseret Alphabet is shown in *Table 13-6*.

Table 13-6. IPA Transcription of Deseret

ᄁᄁ	LONG I	i	ᄃᄃ	BEE	b
ᄂᄂ	LONG E	e	ᄄᄄ	TEE	t
ᄃᄃ	LONG A	a	ᄅᄅ	DEE	d
ᄄᄄ	LONG AH	ɒ	ᄆᄆ	CHEE	tʃ
ᄅᄅ	LONG O	o	ᄇᄇ	JEE	dʒ
ᄆᄆ	LONG OO	u	ᄈᄈ	KAY	k
ᄇᄇ	SHORT I	ɪ	ᄉᄉ	GAY	g
ᄈᄈ	SHORT E	ɛ	ᄊᄊ	EF	f
ᄉᄉ	SHORT A	æ	ᄋᄋ	VEE	v
ᄊᄊ	SHORT AH	ɔ	ᄌᄌ	ETH	θ
ᄋᄋ	SHORT O	ʌ	ᄍᄍ	THEE	ð
ᄌᄌ	SHORT OO	ʊ	ᄎᄎ	ES	s
ᄍᄍ	AY	aɪ	ᄏᄏ	ZEE	z
ᄎᄎ	OI	ɔɪ	ᄐᄐ	ESH	ʃ
ᄏᄏ	OW	aʊ	ᄑᄑ	ZHEE	ʒ
ᄐᄐ	EW	ju	ᄒᄒ	ER	r
ᄑᄑ	WU	w	ᄓᄓ	EL	l
ᄒᄒ	YEE	j	ᄔᄔ	EM	m
ᄓᄓ	H	h	ᄕᄕ	EN	n
ᄔᄔ	PEE	p	ᄌᄌ	ENG	ŋ

13.11 Shavian

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 `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.

13.12 Lisu

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 [a] 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 *Table 13-7*, with the tones identified in terms of their pitch contours.

Table 13-7. 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. Furthermore, each tone letter is typeset on an em-square, including those whose visual appearance consists of two marks.

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.

Other Modifier Letters. Nasalised 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 [a] 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 SEMI-COLON 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 13-8* lists all known adopted punctuation, along with the respective contexts of use.

Table 13-8. 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: “...=”

Linebreaking. 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. There is no line-breaking hyphenation of words, except in 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.

13.13 Miao

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 13.9*,

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.

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).

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.

Chapter 14

Additional Ancient and Historic Scripts

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. A few, such as Coptic, also have contemporary use for liturgical or other special purposes. Some of the historic scripts are related to each other as well as to modern alphabets.

The following ancient and historic scripts are encoded in this version of the Unicode Standard and described in this chapter:

<i>Ogham</i>	<i>Ancient Anatolian Alphabets</i>	<i>Avestan</i>
<i>Old Italic</i>	<i>Old South Arabian</i>	<i>Ugaritic</i>
<i>Runic</i>	<i>Phoenician</i>	<i>Old Persian</i>
<i>Gothic</i>	<i>Imperial Aramaic</i>	<i>Sumero-Akkadian</i>
<i>Old Turkic</i>	<i>Mandaic</i>	<i>Egyptian Hieroglyphs</i>
<i>Linear B</i>	<i>Inscriptional Parthian</i>	<i>Meroitic</i>
<i>Cypriot Syllabary</i>	<i>Inscriptional Pahlavi</i>	

The following ancient and historic scripts are also encoded in this version of the Unicode Standard, but are described in other chapters for consistency with earlier versions of the Unicode Standard, and due to their close relationship with other scripts described in those chapters:

Coptic *Glagolitic* *Phags-pa* *Kaithi* *Kharoshthi* *Brahmi*

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.

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. Old Italic had other descendants, too: The Alpine alphabets seem to have been influential in devising the Runic script, which has a distinct angular appearance owing to its use in carving inscriptions in stone and wood. Gothic, like Cyrillic, was developed on the basis of Greek at a much later date than Old Italic.

The two historic scripts of northwestern Europe, Runic and Ogham, have a distinct appearance owing to their primary use in carving inscriptions in stone and wood. They are con-

ventionally rendered from left to right in scholarly literature, but on the original stone carvings often proceeded in an arch tracing the outline of the stone.

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 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.

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.

The elegant 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. Used over a wide region beginning in the eighth century BCE as Aramaic became the principal administrative language of the Assyrian empire and then the official language of the Achaemenid Persian empire, it was the source of many other scripts, such as the square Hebrew script and the Arabic script. The Mandaic script was probably derived from a cursive form of Aramaic, and was used in southern Mesopotamia for liturgical texts by adherents of the Mandaean gnostic religion. Inscriptional Parthian, Inscriptional Pahlavi, and Avestan are also derived from Imperial Aramaic, and were used to write various Middle Persian languages.

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.

14.1 Ogham

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 (*fedá*), the first 20 of which are considered to be primary and the last 6 (*forfedá*) 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*, *Eabhadh*, *Ór*, and *Ifin*, 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).

Forfedá (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* >π+⋈+⋈+⋈+⋈+⋈+⋈+⋈+⋈+⋈< 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.

14.2 Old Italic

Old Italic: U+10300–U+1032F

The Old Italic script unifies a number of related historical alphabets located on the Italian peninsula. Some of these were used for non-Indo-European languages (Etruscan 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 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. Faliscan, Oscan, Umbrian, North Picene, and South Picene all derive from an Etruscan form of the alphabet.

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; and South Picene adds LETTER II and LETTER UU.

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. The unified Alpine script, which includes the Venetic, Rhaetic, Lepontic, and Gallic alphabets, has not yet been proposed for addition to the Unicode Standard but is considered to differ enough from both Old Italic and Latin to warrant independent encoding. The Alpine script is thought to be the source for Runic, which is encoded at U+16A0..U+16FF. (See *Section 14.3, Runic.*)

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:] became [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 early Etruscan texts have right-to-left directionality. From the third century BCE, left-to-right texts appear, showing the influence of Latin. Oscan, Umbrian, and Faliscan 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.

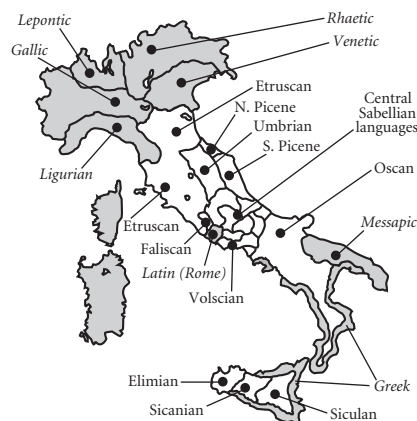
Punctuation. The earliest inscriptions are written with no space between words in what is called *scriptio continua*. There are numerous Etruscan inscriptions with dots separating word forms, attested as early as the second quarter of the seventh century BCE. This punctuation is sometimes, but only rarely, used to separate syllables rather than 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 abecedyary (mid-seventh century BCE). Note that the phonetic values for U+10317 OLD ITALIC LETTER EKS [ks] and U+10319 OLD ITALIC LETTER KHE [kh] show the influence of western, Euboean Greek; eastern Greek has U+03A7 GREEK CAPITAL LETTER CHI [x] and U+03A8 GREEK CAPITAL LETTER PSI [ps] instead.

The geographic distribution of the Old Italic script is shown in *Figure 14-1*. 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 oblique type. In particular, note that the ancient Greek colonies of the southern Italian and Sicilian coasts used the Greek script proper. Also, languages such as Ligurian, Venetic, and so on, of the far north of Italy made use of alphabets of the Alpine script. Rome, of course, is shown in gray, because Latin was written with the Latin alphabet, now encoded in the Latin script.

Figure 14-1. Distribution of Old Italic



14.3 Runic

Runic: U+16A0–U+16F0

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.

Historical Script. 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.

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.

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:

Ʊ ᚋ ᚔ ᚕ ᚖ ᚗ ᚘ ᚙ ᚛ ᚜ ᚝ ᚞ ᚟ ᚠ ᚡ ᚢ ᚣ ᚤ ᚥ ᚦ ᚧ ᚨ ᚩ ᚪ ᚫ

They are usually transliterated in this way:

f u þ a r k g w h n i j ð 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:

Ʊ ᚋ ᚔ ᚕ ᚖ ᚗ ᚘ ᚙ ᚛ ᚜ ᚝ ᚞ ᚟ ᚠ ᚡ ᚢ ᚣ ᚤ ᚥ ᚦ ᚧ ᚨ ᚩ ᚪ ᚫ

f u þ o r k h n i a s t b m l r

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.

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-Frisian 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 Ʊ RUNIC LETTER GER is actually a rare form, as the more common form is already used for U+16E1 Ʊ 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 sys-

tems 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. A total of 81 characters of the Runic script are included in the Unicode Standard. Of these, 75 are Runic letters, 3 are punctuation marks, and 3 are Runic symbols. 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.

14.4 Gothic

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:

SYE ƆAMELIƆ İST İN ESÄİIN PRAUFETAU
swe gameliþ ist in esaïin praufetau
 “as is written in Isaiah the prophet”

To indicate contractions or omitted letters, U+0305 COMBINING OVERLINE is used.

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 one side 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:

•Ḗ or Ḗ̅ 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.

14.5 Old Turkic

Old Turkic: U+10C00–U+10CAF

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 Uighur 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.

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 90° 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 indicated 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.

14.6 Linear B

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.

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 Ṁ LINEAR B IDEOGRAM B100 MAN. In other instances, the names of the ideograms simply carry their catalog number.

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 digits appear 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.

14.7 Cypriot Syllabary

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 14-1*. 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 14-1. Similar Characters in Linear B and Cypriot

Linear B	Cypriot
da 𐀄	ta 𐀄
na 𐀅	na 𐀅
pa 𐀆	pa 𐀆
ro 𐀇	lo 𐀇
se 𐀈	se 𐀈
ti 𐀉	ti 𐀉
to 𐀊	to 𐀊

For Aegean numbers, see the subsection “Aegean Numbers: U+10100–U+1013F” in *Section 14.6, Linear B*.

14.8 Ancient Anatolian Alphabets

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 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.

14.9 Old South Arabian

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-

Sabaeen, Sabaeen 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 Sabaeen 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 13.1, 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 diacritic 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 “1” 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 𐩦 OLD SOUTH ARABIAN NUMERIC INDICATOR before and after the number. Six characters have numeric values as shown in *Table 14-2*—four of these are letters that double as numeric values, and two are characters not used as letters.

Table 14-2. Old South Arabian Numeric Characters

Code Point	Glyph	Numeric function	Other function
10A7F	𐩦	numeric separator	
10A7D		1	word separator
10A6D	𐩣	5	kheth
10A72	𐩠	10	ayn
10A7E	𐩡	50	
10A63	𐩢	100	mem
10A71	𐩠	1000	alef

Numbers are built up through juxtaposition of these characters in a manner similar to that of Roman numerals, as shown in *Table 14-3*. When 10, 50, or 100 occur preceding 1000

they serve to indicate multiples of 1000. The example numbers shown in *Table 14-3* are rendered in a right-to-left direction in the last column.

Table 14-3. Number Formation in Old South Arabian

Value	Schematic	Character Sequence	Display
1	1	10A7D	𐩀
2	1 + 1	10A7D 10A7D	𐩀𐩀
3	1 + 1 + 1	10A7D 10A7D 10A7D	𐩀𐩀𐩀
5	5	10A6D	𐩁
7	5 + 1 + 1	10A6D 10A7D 10A7D	𐩁𐩀𐩀
16	10 + 5 + 1	10A72 10A6D 10A7D	𐩀𐩁𐩀
1000	1000	10A71	𐩀𐩀
3000	1000 + 1000 + 1000	10A71 10A71 10A71	𐩀𐩀𐩀
10000	10 × 1000	10A72 10A71	𐩀𐩀𐩀
11000	10 × 1000 + 1000	10A72 10A71 10A71	𐩀𐩀𐩀𐩀
30000	(10 + 10 + 10) × 1000	10A72 10A72 10A72 10A71	𐩀𐩀𐩀𐩀
30001	(10 + 10 + 10) × 1000 + 1	10A72 10A72 10A72 10A71 10A7D	𐩀𐩀𐩀𐩀𐩀

Names. Character names are based on those of corresponding letters in northwest Semitic.

14.10 Phoenician

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 forerunner 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 Palaeo-Hebrew, Archaic Phoenician, Phoenician, Early Aramaic, Late Phoenician cursive, Phoenician papyri, Siloam Hebrew, Hebrew seals, Ammonite, Moabite, and Punic. The line from Phoenician to Punic is taken to constitute a single continuous branch of script evolution, distinct from that of other related but separately encoded Semitic scripts.

The earliest Hebrew language texts were written in the Palaeo-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 Palaeo-Hebrew forms, eventually developing them into the distinct Samaritan script. (See *Section 8.4, Samaritan.*) The Jews in exile gave up the Palaeo-Hebrew alphabet and instead adopted Imperial Aramaic writing, which was a descendant of the Early Aramaic form of the Phoenician script. (See *Section 14.11, Imperial Aramaic.*) Later, they transformed Impe-

rial Aramaic into the “Jewish Aramaic” script now called (Square) Hebrew, separately encoded in the Hebrew block in the Unicode Standard. (See *Section 8.1, Hebrew*.)

Some scholars conceive of the language written in the Palaeo-Hebrew form of the Phoenician script as being quintessentially Hebrew and consistently transliterate it into Square Hebrew. In such contexts, Palaeo-Hebrew texts are often considered to simply *be* Hebrew, and because the relationship between the Palaeo-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: $\text{|||}\text{𐤁}\text{𐤂}$ 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.

Names. The names used for the characters here are those reconstructed by Theodor Nöldeke in 1904, as given in Powell (1996).

14.11 Imperial Aramaic

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 14-4*. The example numbers shown in the last column are rendered in a right-to-left direction.

Table 14-4. Number Formation in Aramaic

Value	Schematic	Character Sequence	Display
1	1	10858	𐤀
2	2	10859	𐤁
3	3	1085A	𐤂
4	3 + 1	1085A 10858	𐤂𐤀
5	3 + 2	1085A 10859	𐤂𐤁
9	3 + 3 + 3	1085A 1085A 1085A	𐤂𐤂𐤂
10	10	1085B	𐤃
11	10 + 1	1085B 10858	𐤃𐤀
12	10 + 2	1085B 10859	𐤃𐤁
20	20	1085C	𐤄
30	20 + 10	1085C 1085B	𐤄𐤃
55	20 + 20 + 10 + 3 + 2	1085C 1085C 1085B 1085A 10859	𐤄𐤄𐤃𐤂𐤁
70	20 + 20 + 20 + 10	1085C 1085C 1085C 1085B	𐤄𐤄𐤄𐤃
100	1 × 100	10858 1085D	𐤀𐤅
200	2 × 100	10859 1085D	𐤁𐤅
500	(3 + 2) × 100	1085A 10859 1085D	𐤂𐤁𐤅
3000	3 × 1000	1085A 1085E	𐤂𐤅
30000	3 × 10000	1085A 1085F	𐤂𐤆

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 and Inscriptional Pahlavi scripts use a similar system for forming numeric values.

14.12 Mandaic

Mandaic: U+0840—U+085F

The origins of the Mandaic script are unclear, but it is thought to have evolved between the 2nd and 7th 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 Man-

daean 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.

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.

Two letters have morphemic function. U+0847 MANDAIC LETTER IT is used only for the third person singular suffix. U+0856 MANDAIC LETTER DUSHENNA, also called *adu*, is used to write the relative pronoun and the genitive exponent *di*, and 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+0857 MANDAIC LETTER KAD is derived from an old ligature of *ak* + *dushenna*; it is a digraph used to write the word *kd*, which means “when, as, like”. The second additional character, U+0858 MANDAIC LETTER AIN, is a borrowing from U+0639 ARABIC LETTER AIN.

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 14-5* and *Table 14-6*. In these tables, X_n , X_r , X_m , and X_l designate the

Table 14-5. Dual-Joining Mandaic Characters

Character	X_n	X_r	X_m	X_l
AB				
AG				
AD				
AH				
USHENNA				

Table 14-5. Dual-Joining Mandaic Characters (Continued)

Character	X _n	X _r	X _m	X _l
IT	𐤢	𐤢	𐤢	𐤢
ATT	𐤣	𐤣	𐤣	𐤣
AK	𐤤	𐤤	𐤤	𐤤
AL	𐤥	𐤥	𐤥	𐤥
AM	𐤦	𐤦	𐤦	𐤦
AN	𐤧	𐤧	𐤧	𐤧
AS	𐤨	𐤨	𐤨	𐤨
AP	𐤩	𐤩	𐤩	𐤩
ASZ	𐤪	𐤪	𐤪	𐤪
AQ	𐤫	𐤫	𐤫	𐤫
AR	𐤬	𐤬	𐤬	𐤬
AT	𐤭	𐤭	𐤭	𐤭

nominal, right-joining, dual-joining (medial), and left-joining forms respectively, just as in Table 8-7, Table 8-8, and Table 8-9.

Table 14-6. Right-Joining Mandaic Characters

Character	X _n	X _r
HALQA	𐤮	𐤮
AZ	𐤯	𐤯
AKSA	𐤰	𐤰
IN	𐤱	𐤱
ASH	𐤲	𐤲

Linebreaking. 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.

14.13 Inscriptional Parthian and Inscriptional Pahlavi

Inscriptional Parthian: U+10B40–U+10B5F

Inscriptional Pahlavi: U+10B60–U+10B7F

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 Sassanid Persian 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. Other varieties of Pahlavi script include Psalter Pahlavi and the later Book Pahlavi.

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. Conformance 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 *nuns* will nest, and the tail of a *nun* that precedes a *daleth* may be displayed between the two parts of the *daleth* glyph. Table 14-7 shows these behaviors.

Table 14-7. Inscriptional Parthian Shaping Behavior

Character Sequence	Glyph Sequence	Resulting Display	Transcription
<i>gimel-waw</i>	𐎠 𐎡	𐎠𐎡	gw
<i>heth-waw</i>	𐎢 𐎡	𐎢𐎡	xw
<i>yodh-waw</i>	𐎣 𐎡	𐎣𐎡	yw
<i>nun-waw</i>	𐎤 𐎡	𐎤𐎡	nw
<i>ayin-lamedh</i>	𐎥 𐎦	𐎥𐎦	‘l
<i>resh-waw</i>	𐎧 𐎡	𐎧𐎡	rw
<i>taw-waw</i>	𐎨 𐎡	𐎨𐎡	tw
<i>nun-nun</i>	𐎤 𐎤	𐎤𐎤	nn
<i>nun-daleth</i>	𐎤 𐎥	𐎤𐎥	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 they way numbers are built up for Imperial Aramaic; see Table 14-4. 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”.

14.14 Avestan

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 pur-

poses. 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 14-8. U+200C ZERO WIDTH NON-JOINER can be used to prevent ligature formation.

Table 14-8. Avestan Shaping Behavior

Character Sequence	Display	Transcription
<10B31 𐬀 she, 10B00 𐬀 a>	𐬀𐬀	ša
<10B31 𐬀 she, 10B17 𐬀 ce>	𐬀𐬀𐬀	šc
<10B31 𐬀 she, 10B19 𐬀 te>	𐬀𐬀𐬀	št
<10B00 𐬀 a, 10B35 𐬀 he>	𐬀𐬀	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 AVESTAN ABBREVIATION MARK is used to mark abbreviation and repetition. U+10B3D LARGE ONE DOT OVER TWO DOTS PUNCTUATION and U+10B3F LARGE ONE RING OVER TWO RINGS PUNCTUATION 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 WORD SEPARATOR MIDDLE DOT, U+00B7 MIDDLE DOT, or U+002E FULL STOP can be used to represent this.

14.15 Ugaritic

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 Neo-

lithic 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 𐎁 UGARITIC LETTER I, U+1039C 𐎂 UGARITIC LETTER U, and U+1039D 𐎃 UGARITIC LETTER SSU are used for Hurrian.

Variant Glyphs. There is substantial variation in glyph representation for Ugaritic. Glyphs for U+10398 𐎄 UGARITIC LETTER THANNA, U+10399 𐎅 UGARITIC LETTER GHAIN, and U+1038F 𐎆 UGARITIC LETTER DHAL differ somewhat between modern reference sources, as do some transliterations. U+10398 𐎄 UGARITIC LETTER THANNA is most often displayed with a glyph that looks like an occurrence of U+10393 𐎃 UGARITIC LETTER AIN overlaid with U+10382 𐎆 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.

14.16 Old Persian

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 𐎎 OLD PERSIAN SIGN LA, was actually borrowed. It was derived from the New Assyrian historic variant 𐎎 of Sumero-Akkadian U+121B7 𐎎 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.

14.17 Sumero-Akkadian

Cuneiform: U+12000–U+123FF

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 (northwestern 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 \oplus . 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, Assyrian, Eblaite), Elamite, Hittite, and Hurrian.

Periods of script usage are defined according to geography and primary linguistic representation, as shown in *Table 14-9*.

Table 14-9. Cuneiform Script Usage

Archaic Period (to 2901 BCE)		Elamite (2100–360 BCE)
Early Dynastic (2900–2335 BCE)		
Old Akkadian (2334–2154 BCE)		
Ur III (NeoSumerian) (2112–2095 BCE)		
Old Assyrian (1900–1750 BCE)	Old Babylonian (2004–1595 BCE)	
Middle Assyrian (1500–1000 BCE)	Middle Babylonian (1595–627 BCE)	
Neo-Assyrian (1000–609 BCE)		
	Neo-Babylonian (626–539 BCE)	
Hittite (1570–1220 BCE)		

Sources and Coverage. The base character repertoire for the Cuneiform block was distilled from the list of Ur III signs compiled by the Cuneiform Digital Library Initiative (UCLA) in union with the list constructed independently by Miguel Civil. This repertoire is comprehensive from the Ur III period onward. Old Akkadian, Early Dynastic, and Archaic Cuneiform are not covered by this repertoire.

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 or 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, an 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.

Fonts for the representation of Cuneiform text may need to be designed distinctly for optimal use for different historic periods. Fonts for some periods will contain duplicate glyphs depending on the status of merged or split signs at that point of the development of the writing system.

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.

Ordering. The characters are encoded in the Unicode Standard in Latin alphabetical order by primary sign name. Complex signs based on the primary sign are organized according to graphic principles; in some cases, these correspond to the native analyses.

Other Standards. There is no standard legacy encoding of Cuneiform primarily because it was not possible to encode the huge number of characters in the pre-Unicode world of 8-bit fonts.

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).

14.18 Egyptian Hieroglyphs

Egyptian Hieroglyphs: U+13000–U+1342F

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 hiero-

glyphs were in regular use. During the Greco-Roman period, the number of variants, as distinguished by some modern scholars, grew to somewhere between 6,000 and 8,000.

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 hieroglyphs made use of 24 letters comprising a true alphabet. In addition to these phonetic characters, Egyptian hieroglyphs made use of a very large number of logographic characters (called “logograms” or “ideograms” by Egyptologists), some of which could be read as a word, and some of which had only a semantic determinative function, to enable the reader to distinguish between words which were otherwise written the same. Within a word, characters were arranged together to form an aesthetically-pleasing arrangement within a notional square.

Directionality. Characters may be written left-to-right or right-to-left, generally in horizontal lines, but often—especially in monumental texts—in vertical columns. 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.

Egyptian hieroglyphs are given strong left-to-right directionality in the Unicode Standard, because most Egyptian editions are published in English, French, or German, and left-to-right directionality is the conventional 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. A higher-level protocol is required to represent the arrangement of signs into notional squares and for effects involving mirroring or rotation of signs within text. This approach to encoding the hieroglyphs works well in the context of pre-existing conventions for the representation of Egyptian text, which use simple markup schemes to indicate such formatting.

The most prominent example of such conventions in use since computers were introduced into Egyptology in the late 1970s and the early 1980s is called the *Manuel de Codage* (MdC), published in 1988. The MdC conventions make use of ASCII characters to separate hieroglyphic signs and to indicate the organization of the elements in space—that is, the position of each sign, as arranged in a block. For example, the hyphen-minus “-” is used to separate adjacent hieroglyphic blocks. The colon “:” indicates the superposition of one hieroglyphic sign over another. The asterisk “*” indicates the left-right juxtaposition of two hieroglyphic signs within a visual block.

For example, using the MdC conventions, the hieroglyphic representation of the name Amenhotep would be transliterated as <i-mn:n-R4:t*p>. The lowercase letters represent transliterations of alphabetic or other phonetic signs, whereas “R4” is the catalog label for one of the logograms in the standard Gardiner list. The “-”, “:”, and “*” characters provide the markup showing how the individual signs are visually arranged. The “<” and “>” bracket characters indicate a cartouche, often used for a king’s name. The Unicode representation of the same hieroglyphic name, using MdC conventions, but substituting Unicode characters for the transliterations and catalog numbers is shown in *Table 14-10*.

Table 14-10. Hieroglyphic Character Sequence

U+003C	LESS-THAN SIGN
U+131CB	EGYPTIAN HIEROGLYPH M017 (= y, i)
U+002D	HYPHEN-MINUS
U+133E0	EGYPTIAN HIEROGLYPH Y005 (= mn)
U+003A	COLON
U+13216	EGYPTIAN HIEROGLYPH N035 (= n)
U+002D	HYPHEN-MINUS
U+132B5	EGYPTIAN HIEROGLYPH R004 (= R4)
U+003A	COLON
U+133CF	EGYPTIAN HIEROGLYPH X001 (= t)
U+002A	ASTERISK
U+132AA	EGYPTIAN HIEROGLYPH Q003 (= p)
U+003E	GREATER-THAN SIGN

The interpretation of these MdC markup conventions in text is not part of plain text. Ordinary word processors and plain text display would not be expected to be able to interpret those conventions to render sequences of Egyptian hieroglyphic signs stacked correctly into blocks. Instead, such display would require a specialized rendering process familiar with the layout of Egyptian hieroglyphs. This distinction is illustrated in *Figure 14-2*. The first line shows the marked-up MdC sequence for the name of the king, Amenhotep. The second line shows the Unicode hieroglyphic version of that sequence, as interpreted by an ordinary Unicode plain text rendering process. The third line shows a rendering by a specialized hieroglyphic rendering process, which can interpret the markup and render a cartouche.

Figure 14-2. Interpretation of Hieroglyphic Markup

Manuel de Codage: <i-mn:n-R4:t* p>

Unicode Plain Text: (𐀀-𐀁:𐀂-𐀃:𐀄*𐀅)

Interpreted Markup: (𐀀𐀁𐀂𐀃𐀄𐀅)

Other markup schemes have been proposed, which attempt to provide greater flexibility than MdC by use of more elaborate encodings. XML has also been used to represent Egyptian texts. Such representations also require specialized rendering systems to lay out hieroglyphic text.

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. In principle a hieratic font could be devised for specialist applications, but as for fonts for other cursive writing systems, it would require very large ligature tables—even larger than usual, because of the great many hieroglyphic signs involved.

Repertoire. The set of hieroglyphic characters encoded in this 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 the Unicode Standard make use of the standard Egyptological catalog numbers for the signs. Thus, the name for U+13049 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 SIGN G007, U+13147 EGYPTIAN SIGN G007A, and U+13148 EGYPTIAN SIGN 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 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 a coil of rope, as shown in *Figure 14-2*.

In the Unicode representation of hieroglyphic text, the beginning and end of the cartouche are represented by separate paired characters, somewhat like parentheses. Rendering of a full cartouche surrounding a name requires specialized layout software.

There are a several characters for these start and end cartouche characters, reflecting various styles for the enclosures.

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 *Manual*

de Codage, and is comprised of five basic text elements used to build up Egyptian numerals. There is some overlap between these two systems.

14.19 Meroitic Hieroglyphs and Meroitic Cursive

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 14.18, 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 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.

Chapter 15

Symbols

The universe of symbols is rich and open-ended. The collection of encoded symbols in the Unicode Standard encompasses the following:

<i>Currency symbols</i>	<i>Geometrical symbols</i>
<i>Letterlike symbols</i>	<i>Miscellaneous symbols and dingbats</i>
<i>Mathematical alphabets</i>	<i>Emoticons</i>
<i>Numerals</i>	<i>Enclosed and square symbols</i>
<i>Superscript and subscript symbols</i>	<i>Braille patterns</i>
<i>Mathematical symbols</i>	<i>Western and Byzantine musical symbols</i>
<i>Invisible mathematical operators</i>	<i>Ancient Greek musical notation</i>
<i>Technical symbols</i>	

There are other notational systems not covered by the Unicode Standard. Some symbols mark the transition between pictorial items and text elements; because they do not have a well-defined place in plain text, they are not encoded here.

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.

Braille patterns are a special case, because they can be used to write text. They are included as symbols, as the Unicode Standard encodes only their shapes; the association of letters to patterns is left to other standards. When a character stream is intended primarily to convey text information, it should be coded using one of the scripts. Only when it is intended to convey a particular binding of text to Braille pattern sequence should it be coded using 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, and it is primarily those symbols that are encoded in the Unicode Standard. Additional sets of symbols are encoded to support historical systems of musical notation.

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 compatibility with the core *emoji* (“picture character,” or pictograph) sets encoded by several Japanese cell phone carriers as extensions of the JIS X 0208 character set. Those symbols are interchanged as plain text, and are encoded in the Unicode Standard to support interoperability with data originating from the Japanese cell phone carriers. Newer emoji-like symbols are still being developed for mobile phones in Japan, China, and elsewhere, but those pictographs are represented and interchanged using other technologies such as embedded graphics, rather than as plain text. Hence there is no requirement to encode them as characters.

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.”

15.1 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 *Section B.6, Other Unicode Online Resources*. 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 15-1*.

Figure 15-1. 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.

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 15-1*.

Table 15-1. 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
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+FD9C	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 should be used for both the various currencies known as pound (or punt) and the various currencies known as lira—for example, the former currency of Italy and the lira still in use in Turkey. Widespread implementation practice in Italian and Turkish systems has long made use of U+00A3 as the currency sign for the lira. 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.

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. While there may be some preferences to use a double-bar glyph for the yen currency of Japan (JPY) and a single-bar glyph for the yuan (renminbi) currency of China (CNY), this distinction is not systematic in all font designs,

and there is considerable overlap in usage. As listed in *Table 15-1*, 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. 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, with details of glyphic presentation left to font choice and local preferences.

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 15-1* 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.

For additional forms of currency symbols, see Fullwidth Forms (U+FFE0..U+FFE6). Ancient Roman coin symbols, for such coins and values as the *denarius* and *as*, are encoded in the Ancient Symbols block (U+10190..U+101CF).

15.2 Letterlike Symbols

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 15-2* 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 15-2. Alternative Glyphs for Numero Sign

№ № N^o N^o № №

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 12.1, 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 15.10, 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.

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.

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 ϰ, 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 HIRAGANA 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 Figure 15-3.

Figure 15-3. 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_EX 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_EX. 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 *Figure 15-4*.

Figure 15-4. Style Variants and Semantic Distinctions in Mathematics

$$\begin{aligned} \text{Hamiltonian formula:} \quad \mathcal{H} &= \int d\tau (\epsilon E^2 + \mu H^2) \\ \text{Integral equation:} \quad H &= \int d\tau (\epsilon E^2 + \mu H^2) \end{aligned}$$

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 alphanumeric symbols are listed in *Table 15-2*.

Table 15-2. 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
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 math styles in *Table 15-2* 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 in this block encode only semantic distinction, but not which specific font will be used to supply the actual plain, script, Fraktur, double-struck, sans-serif, or monospace glyphs. Especially the script and double-struck styles can show considerable variation across fonts. Characters from the Mathematical Alphanumeric Symbols block are not to be used for nonmathematical styled text.

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.”

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 15-5*.

Figure 15-5. Easily Confused Shapes for Mathematical Glyphs

italic a	<i>a</i>	α	alpha
italic v (pointed)	<i>v</i>	ν	nu
italic v (rounded)	<i>υ</i>	υ	upsilon
script X	<i>ℵ</i>	χ	chi
plain Y	Υ	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 **ℒ** and **ℓ**, 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.

Type Style for Script Characters. In some instances, a deliberate unification with a non-mathematical symbol has been undertaken; for example, U+2133 is unified with the pre-1949 symbol for the German currency unit *Mark*. This unification restricts the range of glyphs that can be used for this character in the charts. Therefore the font used for the representative glyphs in the code charts is based on a simplified “English Script” style, as per recommendation by the American Mathematical Society. For consistency, other script characters in the Letterlike Symbols block are now shown in the same type style.

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 serified and non-serified forms can be used in mathematical texts, and inline fonts are found in works published by certain publishers.

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 MATHEMATICAL 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 letter forms.

15.3 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 15.2, 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.

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 15-3* 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 sets of digits.

Table 15-3. Script-Specific Decimal Digits

Script	Section	Script	Section
Arabic*	<i>Section 8.2</i>	Ol Chiki	<i>Section 10.13</i>
Devanagari	<i>Section 9.1</i>	Sora Sompeng	<i>Section 10.14</i>
Bengali (Bangla)	<i>Section 9.2</i>	Brahmi	<i>Section 10.16</i>
Gurmukhi	<i>Section 9.3</i>	Thai	<i>Section 11.1</i>
Gujarati	<i>Section 9.4</i>	Lao	<i>Section 11.2</i>
Oriya	<i>Section 9.5</i>	Myanmar*	<i>Section 11.3</i>
Tamil	<i>Section 9.6</i>	Khmer	<i>Section 11.4</i>
Telugu	<i>Section 9.7</i>	New Tai Lue	<i>Section 11.6</i>

Table 15-3. Script-Specific Decimal Digits (Continued)

Script	Section	Script	Section
Kannada	Section 9.8	Tai Tham*	Section 11.7
Malayalam	Section 9.9	Kayah Li	Section 11.9
Tibetan	Section 10.2	Cham	Section 11.10
Lepcha	Section 10.3	Balinese	Section 11.13
Limbu	Section 10.5	Javanese	Section 11.14
Saurashtra	Section 10.8	Sundanese	Section 11.17
Sharada	Section 10.9	Mongolian	Section 13.2
Takri	Section 10.10	Osmanya	Section 13.3
Chakma	Section 10.11	N'Ko	Section 13.5
Meetei Mayek	Section 10.12	Vai	Section 13.6

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_Digit`. 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.)

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 8.2, 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. 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 4-10*, 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 15-6*, which illustrates two different ways the number 1,234 can be written with CJK ideographs.

Figure 15-6. 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.

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 several sets of compatibility digits in the Unicode Standard. Table 15-4 provides a full list of compatibility digits.

Table 15-4. Compatibility Digits

Description	Code Range(s)	Numeric Type	Decomp Type	Section
Fullwidth digits	FF10..FF19	Decimal_Digit	Wide	Section 12.5
Bold digits	1D7CE..1D7D7	Decimal_Digit	Font	Section 15.2
Double struck	1D7D8..1D7E1	Decimal_Digit	Font	Section 15.2
Monospace digits	1D7F6..1D7FF	Decimal_Digit	Font	Section 15.2
Sans-serif digits	1D7E2..1D7EB	Decimal_Digit	Font	Section 15.2
Sans-serif bold digits	1D7EC..1D7F5	Decimal_Digit	Font	Section 15.2
Superscript digits	2070, 00B9, 00B2, 00B3, 2074..2079	Digit	Super	Section 15.4
Subscript digits	2080..2089	Digit	Sub	Section 15.4
Circled digits	24EA, 2080..2089	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	2780..2788	Digit	None	
Dingbat negative circled sans-serif digits	278A..2792	Digit	None	

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_Digit, 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_Digit. 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.

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 Digit, rather than Decimal_Digit. This is to prevent superscripted expressions like 2³ 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 15-4* 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_Digit. 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 15-7*. 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 15-7. 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

Significant regional glyph variants for the Eastern-Arabic Digits U+06F0..U+06F9 also occur, but are not separately encoded. See *Table 8-2* 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 4-11*, 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.

Non-Decimal Radix Systems

A number of scripts have number systems that are not decimal place-value notations. The following provides descriptions or references to descriptions of these 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 *Section 13.1, Ethiopic*.

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 $\langle 1, 10, 5 \rangle$ 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 14.17, 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 14.6, 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 14.10, Phoenician* and, in particular, the discussion with examples of number formation in *Section 14.11, Imperial Aramaic*.

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 *anti-sigma*.

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 letter forms 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*.

Rumi Numeral Forms: U+10E60–U+10E7E

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.

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: <one, ten-thousand, two, thousand, three, hundred, four, ten, six>. See *Table 4-10* 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 $\perp \text{ ㄗ } \text{ ㄗ } \text{ ㄗ }$, whereas the latter would be $\text{ ㄗ } \perp \text{ ㄗ }$.

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 4-11* 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.

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 $\frac{1}{4}$, as shown in the code charts, and forms with a horizontal division line, such as $\frac{1}{4}$, which are considered to be alternative glyphs for the same

fractions, as shown in *Figure 15-8*. A few other vulgar fraction characters are located in the Latin-1 block in the range U+00BC..U+00BE.

Figure 15-8. Alternate Forms of Vulgar Fractions

$$\frac{1}{4} \quad \frac{1}{4}$$

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*.

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. 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.

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.

15.4 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 “1st” or “DC00₁₆”, 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.

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 15.3, 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`).

15.5 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. 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` `SCRIPT CAPITAL H`, as well as the Hebrew letter *alef* used as the first transfinite cardinal symbol encoded by `U+2135` `ALEF SYMBOL`, see “Letterlike Symbols” and “Mathematical Alphanumeric Symbols” in *Section 15.2, 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 sub-disciplines 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. 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.”

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+2218 \circ 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. For relation symbols, the choice of a vertical or forward-slanting stroke typically seems to be an aesthetic one, but both slants 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 16.4, Variation Selectors*, for more information on some particular variants.

Unifications. Mathematical operators such as *implies* \Rightarrow and *if and only if* \Leftrightarrow 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.

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 Δ INCREMENT, U+220F \prod N-ARY PRODUCT, and U+2211 Σ 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 15.6, 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 15.9, Miscellaneous Symbols.* See also *Section 15.7, 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 “Semantics of 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 15.2, Letterlike Symbols.* For additional information on all the mathematical operators and other symbols, see Unicode Technical Report #25, “Unicode Support for Mathematics.”

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.

Supplemental Mathematical Operators: U+2A00–U+2AFF

The Supplemental Mathematical Operators block contains many additional symbols to supplement the collection of mathematical operators.

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`.

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.

Miscellaneous Symbols and Arrows: U+2B00–U+2B7F

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” in Section 15.8, *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.

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 \Rightarrow 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.

Supplemental Arrows

The Supplemental Arrows-A (U+27F0..U+27FF), Supplemental Arrows-B (U+2900..U+297F), and Miscellaneous Symbols and Arrows (U+2B00..U+2BFF) 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.

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.

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 16.4, *Variation Selectors*.

15.6 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 $3\frac{1}{4}$ which occur frequently in school and engineering texts. To ensure that $3\frac{1}{4}$ means 3 plus $\frac{1}{4}$ —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.

15.7 Technical Symbols

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.

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.

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.

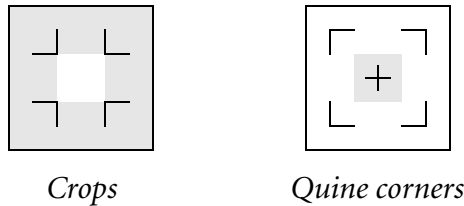
Keytop Labels. Where possible, keytop labels have been unified with other symbols of like appearance—for example, U+21E7 UPWARDS WHITE ARROW to indicate the Shift key. While symbols such as U+2318 PLACE OF INTEREST SIGN and U+2388 HELM SYMBOL are generic symbols that have been adapted to use on keytops, 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 15-9*.

Figure 15-9. Usage of Crops and 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 MATHEMATICAL LEFT ANGLE BRACKET and U+27E9 MATHEMATICAL RIGHT ANGLE BRACKET, should be used instead. See *Section 15.5, 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 relatively 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 Σ 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 15-5 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 15-5. 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*.

Terminal Graphics Characters. In addition to the box drawing characters in the Box Drawing block, a small number of vertical or horizontal line characters are encoded in the Miscellaneous Technical symbols block to complete the set of compatibility characters needed for applications that need to emulate various old terminals. The horizontal scan line characters, U+23BA HORIZONTAL SCAN LINE-1 through U+23BD HORIZONTAL SCAN LINE-9, in particular, 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.

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 15-10*.

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.

Figure 15-10. 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

```

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+23F3 are often found in user interfaces for media players, clocks, alarms, and timers, as well as in text discussing those user interfaces. The Miscellaneous Symbols and Pictographs block also contains many user interface symbols in the ranges U+1F500..U+1F518 and U+1F53A..U+1F53D, 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.

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.

15.8 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.

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 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.

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.

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, and Miscellaneous Symbols 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.

15.9 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 accomodation 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 used in the core *emoji* (“picture character”) set in common use on mobile phones in Japan. The majority of these *emoji* symbols are encoded in the Miscellaneous Symbols and Pictographs and Emoticons blocks, but many *emoji* symbols are encoded in other blocks. For a complete listing of the core *emoji* set, including information about which *emoji* symbols have been unified with other symbol characters in the Unicode Standard, see the data file `EmojiSources.txt` in the Unicode Character Database.

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. For a discussion of the particular issues involved in the interpretation and display of dingbats, see the documentation of the Dingbats block later in this section.

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.

Rendering of Emoji Symbols. Many of the characters in the blocks associated with miscellaneous symbols, in particular the Miscellaneous Symbols and Pictographs, Emoticons, Transport and Map Symbols, and Enclosed Alphanumeric Supplement blocks, are used in the core *emoji* (“picture character”) sets available on cell phones in Japan. Especially in that context, there may be a great deal of variability in presentation, along three axes:

- **Glyph shape:** *Emoji* symbols 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 Japanese cell phone e-mail or text messages) are displayed in color, sometimes as a multicolor image. While this is particularly true of *emoji* symbols, 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 symbols may be presented using color or animation, but need not be. Because many characters in the core *emoji* sets are unified with Unicode characters that originally came from other sources, there is no way based on character code alone to tell whether a character should be presented using an “emoji” style; that decision depends on context.

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 from the core *emoji* sets, other color terms such as BLUE or ORANGE. Neither the monochrome nature of the representative glyph nor any color term in the character name are meant to imply any requirement or limitation on how the glyph may be presented (see also “Images in the Code Charts and Character Lists” in Section 17.1, *Character Names List*). 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 only to help identify the corresponding characters in the core *emoji* sets; the characters may be presented using color, or in monochrome using different styles of shading or crosshatching, for example.

Miscellaneous Symbols: U+2600–U+26FF

Miscellaneous Symbols and Pictographs: U+1F300–U+1F5FF

The Miscellaneous Symbols and the Miscellaneous Symbols and Pictographs 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.)

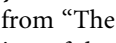
Standards. The symbols in these blocks are derived from a large range of national and vendor character standards. Among them, characters from the Japanese Association of Radio Industries and Business (ARIB) standard STD-B24 are widely represented in the Miscellaneous Symbols block. The symbols from ARIB were initially used in the context of digital broadcasting, but in many cases their usage has evolved to more generic purposes. The core


emoji sets used by Japanese cell phone carriers are another source for a large number of characters in the Miscellaneous Symbols and Pictographs block.



Weather Symbols. The characters in the ranges U+2600..U+2603 and U+26C4..U+26CB, 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 web sites.



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 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 old-style chess 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, 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 core *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 core *emoji* sets. A few of the animal symbols have conventional uses to designate types of meat on menus.

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 the core *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.

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 and the Enclosed Alphanumeric Supplement block U+1F100..U+1F1FF. Some of these blocks contain symbols which extend or complement sets of symbols contained in the Miscellaneous Symbols block.

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 core *emoji* sets used by Japanese cell phone carriers 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.

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. As of Version 6.0, about half of these symbols are from the core *emoji* sets used by Japanese cell phone carriers.

Various traffic signs and map symbols are also encoded in the Miscellaneous Symbols block.

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. Other series of dingbat glyphs also exist, but are not encoded in the Unicode Standard because they are not widely implemented in existing hardware and software as character-encoded fonts. The order of the Dingbats block basically follows the PostScript encoding.

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 core *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.

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.

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-nineteenth century, and its popularity spread to Japan, Britain, and the United States during the early twentieth 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.

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.

Playing Cards: U+1F0A0–U+1F0FF

These characters 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. Also included are a generic card back and two Jokers. U+1F0CF PLAYING CARD BLACK JOKER is used in one of the Japanese cell phone core *emoji* sets; its presentation may be in color and need not be black.

These characters most commonly appear as the Anglo-French-style playing cards used with international bridge or poker. However, in different countries, both the suits and the colors may be substantially different, to the point of being unrecognizable. 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 these characters without additional information (such as a font) or agreement between sender and receiver. Without such information or agreement, the glyphs have only a symbolic and schematic equivalence to particular varieties of actual playing cards.

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 Tarot cards use swords). 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 symbols are encoded in the Miscellaneous Symbols block in the range U+2660..U+2667.

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).

Tai Xuan Jing Symbols: U+1D300–U+1D356

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 MONOGRAM 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).

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

Phaistos Disc Symbols: 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 lay-out 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 14.2, 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.

15.10 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 stacking 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 stacked 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 core *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 stacks of 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 enclosure 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.

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 background 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.

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.

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 *Table 15-6*.

Table 15-6. Japanese Era Names

Code Point	Name	Dates
U+337B	SQUARE ERA NAME HEISEI	1989-01-07 to present day
U+337C	SQUARE ERA NAME SYOUWA	1926-12-24 to 1989-01-06
U+337D	SQUARE ERA NAME TAISYOU	1912-07-29 to 1926-12-23
U+337E	SQUARE ERA NAME MEIZI	1867 to 1912-07-28

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 core *emoji* sets.

Regional Indicator Symbols. The regional indicator symbols in the range U+1F1E6..U+1F1FF can be used in pairs to represent an ISO 3166 region code. This mechanism is not intended to supplant actual ISO 3166 region codes, which simply use Latin letters in the ASCII range; instead the main purpose of such pairs is to provide unambiguous roundtrip mappings to certain characters used in the *emoji* core sets. The representative glyph for region indicator symbols is simply a dotted box containing a letter. The Unicode Standard does not prescribe how the pairs of region indicator symbols should be rendered. In *emoji* contexts, where text is displayed as it would be on a Japanese mobile phone, a pair may be displayed using the glyph for a flag, as appropriate, but in other contexts the pair could be rendered differently. See the file `EmojiSources.txt` in the Unicode Character Database for more information about source mappings involving regional indicator symbols.

Enclosed Ideographic Supplement: U+1F200–U+1F2FF

This block consists mostly of enclosed ideographic symbols. It also contains some additional squared Katakana word symbols. As of Version 6.0, all 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 core *emoji* sets.

The enclosed ideographic symbols in the range U+1F210..U+1F231 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.

15.11 Braille

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:

1	●●	4
2	●●	5
3	●●	6
7	●●	8

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:

1	●●	10
2	●●	20
4	●●	40
100	●●	200

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.”

15.12 Western Musical Symbols

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. 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

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.

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 15-11* 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.

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.

Figure 15-11. Examples of Specialized Music Layout

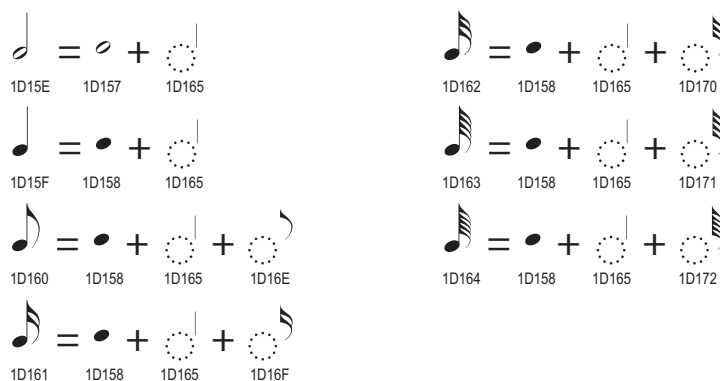


Similarly, format characters have been provided for other connecting structures. The characters U+1D175 MUSICAL SYMBOL BEGIN TIE, U+1D176 MUSICAL SYMBOL END TIE, U+1D177 MUSICAL SYMBOL BEGIN SLUR, U+1D178 MUSICAL SYMBOL END SLUR, U+1D179 MUSICAL SYMBOL BEGIN PHRASE, and U+1D17A MUSICAL SYMBOL END PHRASE 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 15-12.

Figure 15-12. Precomposed Note Characters



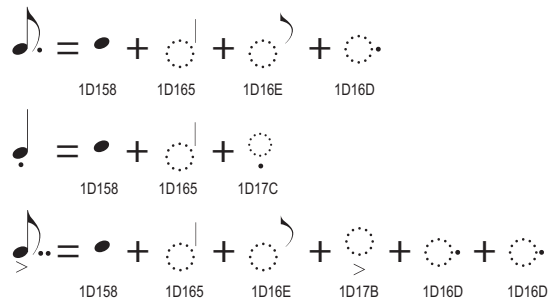
Alternative Noteheads. More complex notes built up from alternative noteheads, stems, flags, and articulation symbols are necessary for complete implementations and complex scores. Examples of their use include American shape-note and modern percussion notations, as shown in Figure 15-13.

Figure 15-13. Alternative Noteheads



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 are shown in Figure 15-14.

Figure 15-14. Augmentation Dots and Articulation Symbols



Ornamentation. Table 15-7 lists common eighteenth-century ornaments and the sequences of characters from which they can be generated.

Table 15-7. 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

15.13 Byzantine Musical Symbols

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 percent 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 15.12, 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.

15.14 Ancient Greek Musical Notation

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 15-8* 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.

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.

Table 15-8. Representation of Ancient Greek Vocal and Instrumental Notation

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

Table 15-8. Representation of Ancient Greek Vocal and Instrumental Notation (Continued)

Modern Note	Modern Number	Vocal Notation	Instrumental Notation
c	17	1D210	1D227
	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

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 serified 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.

Chapter 16

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:

<i>Control codes</i>	<i>Surrogates area</i>	<i>Private-use characters</i>
<i>Layout controls</i>	<i>Variation selectors</i>	<i>Deprecated format characters</i>
<i>Specials</i>	<i>Noncharacters</i>	<i>Deprecated tag characters</i>

In addition to regular characters, the Unicode Standard contains a number of format characters. These characters are not normally rendered directly, but rather influence the layout of text or otherwise affect the operation of text processes.

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 reserved and are to be used in pairs—called surrogate pairs—to access 1,048,544 supplementary characters.

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. These characters are deprecated, and their use is strongly discouraged.

16.1 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 00₁₆ to 1F₁₆ (C0 controls), 7F₁₆ (*delete*), and 80₁₆ to 9F₁₆ (C1 controls), respectively. For example, the 8-bit legacy control code *character tabulation* (or *tab*) is the byte value 09₁₆; 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.

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 (0A₁₆) 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 01₁₆ and ^B refers to the C0 control code 02₁₆, 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, 1B₁₆, 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.

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 16-1* lists those control codes.

Table 16-1. 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

Most of the control codes in *Table 16-1* have the `White_Space` property. They have the `Bidi_Class` property values of S, B, or WS, rather than the default of ON used for other control codes. (See Unicode Standard Annex #9, “Unicode Bidirectional Algorithm.”) In addition, the separator semantics of the control codes U+001C..U+001F are recognized in the Bidirectional Algorithm. U+0009..U+000D and U+0085 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 *Section 16.2, Layout Controls*, 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 *Section 5.8, Newline Guidelines*.

16.2 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.

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 word boundaries; 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 word or line breaking.

Zero Width No-Break Space. In addition to its primary meaning of *byte order mark* (see “Byte Order Mark” in *Section 16.8, 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 charac-

ters separated by a ZWSP. For example, in *Table 16-2*, the row labeled “Display 4” illustrates incorrect suppression of inter-character spacing in the context of a ZWSP.

Table 16-2. Letter Spacing

Type	Justification Examples	Comment
Memory	the ISP [®] Charts	The ^{ZWSP} is inserted to allow line break after *
Display 1	the ISP [®] Charts	Without letter spacing
Display 2	the ISP [®] Charts	Increased letter spacing
Display 3	the ISP [®] Charts	“Thai-style” letter spacing
Display 4	the ISP [®] Charts	^{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 determining 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 indicated 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 10.2, 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 separate 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 paragraph 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 SEPARATOR can be used to indicate an unconditional end of line.

A paragraph separator indicates where a new paragraph should start. Any interparagraph 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 information on line separators, see Section 5.8, *Newline Guidelines*.

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 Unicode Technical Report #20, “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 is intended to produce a more connected rendering of adjacent characters than would otherwise be the case, if possible. 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.

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 16-3*.

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.

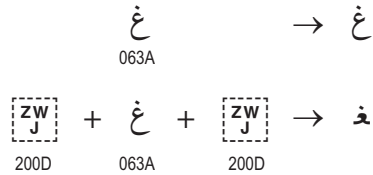
In *Figure 16-1*, the insertion of the ZWNJ overrides the normal cursive joining of *sad* and *lam*.

Figure 16-1. Prevention of Joining

ص	+	ل	→	صل		
0635		0644				
ص	+	ZW NJ	+	ل	→	صل
0635		200C		0644		

In *Figure 16-2*, 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 16-2. Exhibition of Joining Glyphs in Isolation



The examples in *Figure 16-1* and *Figure 16-2* 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 16-3* 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 16-3* 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 16-3. Effect of Intervening Joiners

Character Sequences	As Is	$\boxed{\text{ZW NJ}}$	$\boxed{\text{ZW J}}$ $\boxed{\text{ZW NJ}}$ $\boxed{\text{ZW J}}$	$\boxed{\text{ZW J}}$
f i 0066 0069	f i or fi	fi	fi	fi
ا ل 0627 0644	لا	لا	لا	لا
ج م 062C 0645	مج or مج	مج	مج	مج
و ج 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 ENCLCING 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 8.2, 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 9, South Asian Scripts-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.

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 definition of canonical reordering 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 7-10*.

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` 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.”

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 16-3*.

Table 16-3. Bidirectional Ordering Controls

Code	Name	Abbreviation
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

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.

U+200E LEFT-TO-RIGHT MARK and U+200F RIGHT-TO-LEFT 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.)

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 16-4*.

The bidirectional overrides and embeddings and 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.

Table 16-4. Paired Stateful Controls

Characters	Documentation
Bidi Overrides and Embeddings	Section 16.2, <i>Layout Controls</i> ; UAX #9
Annotation Characters	Section 16.8, <i>Specials</i>
Musical Beams and Slurs	Section 15.12, <i>Western Musical Symbols</i>

Bidirectional overrides and embeddings are default ignorable code points; 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 code points.

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 code points; 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 16.8, *Specials* for more discussion.

Other paired stateful controls in the standard are deprecated, and their use should be avoided. They are listed in Table 16-5.

Table 16-5. Paired Stateful Controls (Deprecated)

Characters	Documentation
Deprecated Format Characters	Section 16.3, <i>Deprecated Format Characters</i>
Tag Characters	Section 16.9, <i>Deprecated Tag Characters</i>

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.

16.3 Deprecated Format Characters

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.


16.4 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 always consists of a base character followed by a variation selector character. That sequence is referred to as a *variant* of the base character.

In a variation sequence the variation selector affects the appearance of the base character. Such changes in appearance may, in turn, have a visual impact on subsequent characters, particularly combining characters applied to that base character. For example, if the base character changes shape, that should result in a corresponding change in shape or position of applied combining marks. If the base character changes color, as can be the case for emoji style variation sequences, the color may also change for applied combining marks. If the base character changes in advance width, that would also change the positioning of subsequent spacing characters.

In particular, the emoji style variation sequences for digits and U+0023 “#” NUMBER SIGN are intended to affect the color, size, and positioning of U+20E3  COMBINING ENCLOSING KEYCAP when applied to those base characters. For example, the variation sequence <0023, FE0F> selects the emoji style 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 variation selector is *not* used as a general code extension mechanism; only certain sequences are defined, as follows:

Standardized variation sequences are defined in the file StandardizedVariants.txt in the Unicode Character Database. 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 two 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 base character from what it would have had in the absence of the variation selector.

The base character in a variation sequence is never a combining character or a decomposable character. The variation selectors themselves are combining marks of combining class 0 and are default ignorable characters. Thus, if the variation sequence is not supported, the variation selector should be invisible and ignored. As with all default ignorable characters,

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 a base 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 base 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.

Mongolian. For the behavior of older implementations of Mongolian using variation selectors, see the discussion of Mongolian free variation selectors in *Section 13.2, Mongolian*.

16.5 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.

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 16.7, 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.

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.

16.6 Surrogates Area

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 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 repre-

sented 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 16.5, Private-Use Characters*.

The code tables do not have charts or name list entries for the range D800..DFFF because individual, unpaired surrogates merely have code points.

16.7 Noncharacters

Noncharacters: U+FFFE, U+FFFF, and Others

Noncharacters are code points that are permanently reserved in the Unicode Standard for internal use. They are forbidden for use in open interchange of Unicode text data. See *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 but should *never* attempt to exchange them. 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 16.5, Private-Use Characters*, which *are* assigned characters and which *are* intended for use in open interchange, subject to interpretation by private agreement, noncharacters are permanently 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, FFFF₁₆. U+10FFFF is associated with the largest legal UTF-32 32-bit code unit value, 10FFFF₁₆. 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 non-character U+FFFE, see *Section 16.8, Specials*.

16.8 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 16.7, Noncharacters*.

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 16.2, 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 $\langle FE_{16} FF_{16} \rangle$ or $\langle FF_{16} FE_{16} \rangle$ 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 $\langle FE_{16} FF_{16} \rangle$ constitutes the string $\langle thorn, y diaeresis \rangle$ “þÿ”; in systems that employ the Apple Macintosh Roman character set or the Adobe Standard Encoding, this sequence represents the sequence $\langle ogonek, hacek \rangle$ “ ˆ ˇ ”; in systems that employ other common IBM PC code pages (for example, CP 437, 850), this sequence represents $\langle black\ square, no\ break\ space \rangle$ “■”.

In UTF-8, the BOM corresponds to the byte sequence $\langle EF_{16} BB_{16} BF_{16} \rangle$. 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,

<EF₁₆ BB₁₆ BF₁₆> corresponds to the sequence <*i diaeresis, guillemet, inverted question mark*> “ı » ¿”.

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 16.2, 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 16-6* for a summary of encoding scheme signatures.

Table 16-6. 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.

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 16-7*.

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.

Table 16-7. 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

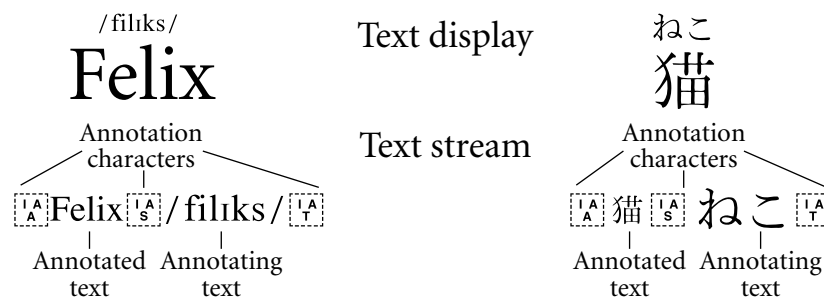
Specials: U+FFF0–U+FFF8

The nine unassigned Unicode code points in the range U+FFF0..U+FFF8 are reserved for special character definitions.

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 characters* delimit the annotating and the annotated text, and identify them as part of an annotation. See Figure 16-4.

Figure 16-4. 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. Anchor characters 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 non-linear—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.)

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.

16.9 Deprecated Tag Characters

Deprecated Tag Characters: U+E000–U+E007F

The characters in this block provide a mechanism for language tagging in Unicode plain text. These characters are deprecated, and should not be used—particularly with any protocols that provide alternate means of language tagging. The Unicode Standard recommends the use of higher-level protocols, such as HTML or XML, which provide for language tagging via markup. See Unicode Technical Report #20, “Unicode in XML and Other Markup Languages.” The requirement for language information embedded in plain text data is often overstated, and markup or other rich text mechanisms constitute best current practice. See *Section 5.10, Language Information in Plain Text* for further discussion.

This block encodes a set of 95 special-use tag characters to enable the spelling out of ASCII-based string tags using characters that can be strictly separated from ordinary text content characters in Unicode. These tag characters can be embedded by protocols into plain text. 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.

In addition to these 95 characters, one language tag identification character and one cancel tag character are encoded. The language tag identification character identifies a tag string as a language tag; the language tag itself makes use of RFC 4646 (or its successors) language tag strings spelled out using the tag characters from this block.

Syntax for Embedding Tags

To embed any ASCII-derived tag in Unicode plain text, the tag is spelled out with corresponding tag characters, prefixed with the relevant tag identification character. The resultant string is embedded directly in the text.

Tag Identification. The tag identification character is used as a mechanism for identifying tags of different types. In the future, this could enable multiple types of tags embedded in plain text to coexist.

Tag Termination. No termination character is required for the tag itself, because all characters that make up the tag are numerically distinct from any non-tag character. A tag terminates either at the first non-tag character (that is, any other normal Unicode character) or at next tag identification character. A detailed BNF syntax for tags is listed in “Formal Tag Syntax” later in this section.

Language Tags. A string of tag characters prefixed by U+E0001 LANGUAGE TAG is specified to constitute a language tag. Furthermore, the tag values for the language tag are to be

Canceling Tag Values. The main function of CANCEL TAG is to make possible operations such as blind concatenation of strings in a tagged context without the propagation of inappropriate tag values across the string boundaries. There are two uses of CANCEL TAG. To cancel a tag value of a particular type, prefix the CANCEL TAG character with the tag identification character of the appropriate type. For example, the complete string to cancel a language tag is <U+E0001, U+E007F>. The value of the relevant tag type returns to the default state for that tag type—namely, no tag value specified, the same as untagged text. To cancel any tag values of any type that may be in effect, use CANCEL TAG without a prefixed tag identification character.

Currently there is no observable difference in the two uses of CANCEL TAG, because only one tag identification character (and therefore one tag type) is defined. Inserting a bare CANCEL TAG in places where only the language tag needs to be canceled could lead to unanticipated side effects if this text were to be inserted in the future into a text that supports more than one tag type.

Working with Language Tags

Avoiding Language Tags. Because of the extra implementation burden, language tags should be avoided in plain text unless language information is required and the receivers of the text are certain to properly recognize and maintain the tags. However, where language tags must be used, implementers should consider the following implementation issues involved in supporting language information with tags and decide how to handle tags where they are not fully supported. This discussion applies to any mechanism for providing language tags in a plain text environment.

Higher-Level Protocols. Language tags should be avoided wherever higher-level protocols, such as a rich text format, HTML, or MIME, provide language attributes. This practice prevents cases where the higher-level protocol and the language tags disagree. See Unicode Technical Report #20, “Unicode in XML and Other Markup Languages.”

Effect of Tags on Interpretation of Text. Implementations that support language tags may need to take them into account for special processing, such as hyphenation or choice of font. However, the tag characters themselves have no display and do not affect line breaking, character shaping or joining, or any other format or layout properties. Processes interpreting the tag may choose to impose such behavior based on the tag value that it represents.

Display. Characters in the tag character block have no visible rendering in normal text and the language tags themselves are not displayed. This choice may not require modification of the displaying program, if the fonts on that platform have the language tag characters mapped to zero-width, invisible glyphs. For debugging or other operations that must render the tags themselves visible, it is advisable that the tag characters be rendered using the corresponding ASCII character glyphs (perhaps modified systematically to differentiate them from normal ASCII characters). The tag character values have been chosen, however, so that the tag characters will be interpretable in most debuggers even without display support.

Processing. Sequential access to the text is generally straightforward. If language codes are not relevant to the particular processing operation, then they should be ignored. Random access to stateful tags is more problematic. Because the current state of the text depends on tags that appeared previous to it, the text must be searched backward, sometimes all the way to the start. With these exceptions, tags pose no particular difficulties as long as no modifications are made to the text.

Range Checking for Tag Characters. Tag characters are encoded in Plane 14 to support easy range checking. The following C/C++ source code snippets show efficient implemen-

tations of range checks for characters E0000..E007F expressed in each of the three significant Unicode encoding forms. Range checks allow implementations that do not want to support these tag characters to efficiently filter for them.

Range check expressed in UTF-32:

```
if ( ((unsigned) *s) - 0xE0000 <= 0x7F )
```

Range check expressed in UTF-16:

```
if ( ( *s == 0xDB40 ) && ( ((unsigned)*(s+1)) - 0xDC00 <= 0x7F ) )
```

Range check expressed in UTF-8:

```
if ( ( *s == 0xF3 ) && ( *(s+1) == 0xA0 ) &&
      ( ( *(s+2) & 0xFE ) == 0x80 ) )
```

Alternatively, the range checks for UTF-32 and UTF-16 can be coded with bit masks. Both versions should be equally efficient.

Range check expressed in UTF-32:

```
if ( ((*s) & 0xFFFFFFFF80) == 0xE0000 )
```

Range check expressed in UTF-16:

```
if ( ( *s == 0xDB40 ) && ( *(s+1) & 0xDC80) == 0xDC00 )
```

Editing and Modification. Inline tags present particular problems for text changes, because they are stateful. Any modifications of the text are more complicated, as those modifications need to be aware of the current language status and the <start>...<end> tags must be properly maintained. If an editing program is unaware that certain tags are stateful and cannot process them correctly, then it is very easy for the user to modify text in ways that corrupt it. For example, a user might delete part of a tag or paste text including a tag into the wrong context.

Dangers of Incomplete Support. Even programs that do not interpret the tags should not allow editing operations to break initial tags or leave tags unpaired. Unpaired tags should be discarded upon a save or send operation.

Nonetheless, malformed text may be produced and transmitted by a tag-unaware editor. Therefore, implementations that do not ignore language tags must be prepared to receive malformed tags. On reception of a malformed or unpaired tag, language tag-aware implementations should reset the language to NONE and then ignore the tag.

Unicode Conformance Issues

The rules for Unicode conformance for the tag characters are exactly the same as those for any other Unicode characters. A conformant process is not required to interpret the tag characters. If it does interpret them, it should interpret them according to the standard—that is, as spelled-out tags. However, there is no requirement to provide a particular interpretation of the text because it is tagged with a given language. If an application does not interpret tag characters, it should leave their values undisturbed and do whatever it does with any other uninterpreted characters.

The presence of a well-formed tag is no guarantee that the data are correctly tagged. For example, an application could erroneously label French data with a Spanish tag.

Implementations of Unicode that already make use of out-of-band mechanisms for language tagging or “heavy-weight” in-band mechanisms such as XML or HTML will continue to do exactly what they are doing and will ignore the tag characters completely. They may even prohibit their use to prevent conflicts with the equivalent markup.

Formal Tag Syntax

An extended BNF description of the tags specified in this section is given here.

```
tag := language-tag | cancel-all-tag
```

```
language-tag := language-tag-introducer (language-tag-arg  
| tag-cancel)
```

```
language-tag-arg := tag-argument
```

In this rule, tag-argument is constrained to be a valid language identifier according to RFC 4646, with the assumption that the appropriate conversions from tag character values to ASCII are performed before checking for syntactic correctness against RFC 4646. For example, U+E0041 TAG LATIN CAPITAL LETTER A is mapped to U+0041 LATIN CAPITAL LETTER A, and so on.

```
cancel-all-tag := tag-cancel
```

```
tag-argument := tag-character+
```

```
tag-character := [U+E0020 - U+E007E]
```

```
language-tag-introducer := U+E0001
```

```
tag-cancel := U+E007F
```


Chapter 17

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 online 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*. Many scripts are fully contained within a single character 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 set of character names, in machine-readable form, appears in the Unicode Character Database.

An index to distinctive character names can also be found on the Unicode Web site.

For information about access to the code charts and the character name index, see *Section B.6, Other Unicode Online Resources*.

17.1 Character Names List

The following illustration identifies the components of typical entries in the character names list.

<i>code</i>	<i>image</i>	<i>entry</i>	
00AE	®	REGISTERED SIGN = registered trade mark sign (1.0)	(Version 1.0 name)
00AF	-	MACRON = overline, APL overbar • this is a spacing character → 02C9 ¯ modifier letter macron → 0304 ¯ combining macron → 0305 ¯ combining overline ≈ 0020 ☐ 0304 ¯	(Unicode name) (alternative names) (informative note) (cross reference) (compatibility decomposition)
00E5	å	LATIN SMALL LETTER A WITH RING ABOVE • Danish, Norwegian, Swedish, Walloon ≡ 0061 a 030A ◌	(sample of language use) (canonical decomposition)

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 serified, 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 “a” 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 *d* instead 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 *N* NUMERO SIGN is a fullwidth shape as it would be used in East Asian fonts. In Cyrillic usage, *N* is the universally recognized glyph. See *Figure 15-2*.

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 *ai* 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 characters in the code charts are oriented as they would appear in normal text. This is true regardless of whether the script in question is predominantly laid out in horizontal lines, as for most scripts, or is predominantly laid out in vertical lines, as for Mongolian and Phags-pa. In cases such as Mongolian, this is accom-

plished using specialized chart fonts which show the glyphs correctly oriented, even though the chart production software lays out all glyphs horizontally in their boxes. Note that commercial production fonts for Mongolian do not behave this way; if used with common charting tools, including those for the Unicode code charts, such fonts will show Mongolian glyphs with their images turned 90 degrees counterclockwise.

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.

0020	☐	SPACE • sometimes considered a control code • other space characters: 2000 ☐ - 200A ☐
------	---	---


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	☐	SOFT HYPHEN = discretionary hyphen • commonly abbreviated as SHY
------	---	--


This convention is also used for some graphic characters which are only distinguished by special behavior from another character of the same appearance.

2011	☐	NON-BREAKING HYPHEN → 002D - hyphen-minus → 00AD ☐ soft hyphen ≈ <noBreak> 2010 -
------	---	--


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 16.4, Variation Selectors*).

FE00  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 the consonant-stacking viramas.



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>
 → 00B3³ 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 16.7, Noncharacters*.

FFFF  <not a character>
 • the value FFFF  is guaranteed not to be a Unicode character at all

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.

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*.

Informative Aliases

An informative alias (preceded by =) is an alternate name for a character. Characters may have several aliases, and aliases for different characters are not guaranteed to be unique. Aliases are informative and may be updated. By convention, aliases are in lowercase, except where they contain proper names. Where an alias matches the name of a character in *The Unicode Standard, Version 1.0*, it is listed first, followed by “1.0” in parentheses. Because the formal character names may differ in unexpected ways from commonly used names (for example, PILCROW SIGN = paragraph sign), some aliases may be useful alternate choices for indicating characters in user interfaces. In the Hangul Jamo block, U+1100..U+11FF, the normative short jamo names are given as aliases.

Normative Aliases

A normative character name alias (one preceded by ✖) is a formal, unique, and stable alternate name for a character. Characters are given normative character name aliases in certain cases where there is a defect in the character name. These aliases do not replace the character name, but rather allow users to formally refer to the character without requiring the use of a defective name. Normative character name aliases which provide information about corrections to defective character names are always printed in the character names list. Normative aliases serving other purposes, such as defining abbreviations for characters, may be omitted or may be presented with an alternative symbol to distinguish them. For a definite list, suitable for machine parsing, see, NameAliases.txt in the UCD. For more information, see *Section 4.8, Name*. By convention, normative character aliases are written in uppercase letters.

```
FE18   ≡   PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRACKET
        ✖   PRESENTATION FORM FOR VERTICAL RIGHT WHITE LENTICULAR BRACKET
        •   misspelling of "BRACKET" in character name is a known defect
        ≈   <vertical> 3017
```

Cross References

Cross references (preceded by →) are used to indicate a related character of interest, but without indicating the nature of the relation. Possibilities are a different character of similar appearance or name, the other member of a case pair, or some other linguistic relationship.

Explicit Inequality. The two characters are not identical, although the glyphs that depict them are identical or very close.

```
003A   :   COLON
        → 0589 : armenian full stop
        → 2236 : ratio
```

Other Linguistic Relationships. These relationships include transliterations (such as between Serbian and Croatian), typographically unrelated characters used to represent the same sound, and so on.

```
01C9   lj   LATIN SMALL LETTER LJ
        → 0459 љ cyrillic small letter lje
        ≈ 006C I 006A j
```

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.

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.

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
 • uppercase is “SS”
 • in origin a ligature of 017F f and 0073 s
 → 03B2 β greek small letter beta

For more information about case and case mappings, see *Section 4.2, Case*.

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 â LATIN SMALL LETTER A WITH RING ABOVE
 • Danish, Norwegian, Swedish, Walloon
 ≡ 0061 a 030A ◌̂

212B Å ÅNGSTROM SIGN
 ≡ 00C5 Å latin capital letter a with ring above

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

The following compatibility formatting tags are used in the Unicode Character Database:

	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 ÅNGSTRÖM SIGN can be further decomposed using the canonical mapping for U+00C5 Å LATIN CAPITAL LETTER A WITH RING ABOVE. (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 WORD JOINER to emulate nonbreaking behavior; compatibility decompositions with the <circle> formatting tag do not use U+20DD COMBINING ENCLOSING CIRCLE; and compatibility decompositions with formatting tags <initial>, <medial>, <final>, or <isolate> for explicit positional forms do not use ZWJ or ZWNJ. The one exception is the use of U+2044 FRACTION SLASH to express the <fraction> semantics of compatibility decompositions for vulgar fractions.

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+26FE, 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 characters could be encoded in the future at reserved code points within their ranges. The subheads are subject to change.

17.2 CJK Unified and Compatibility Ideographs

The code charts for CJK ideographs differ significantly from those for other characters in the standard.

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 represented 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 Rapporteur Group (IRG) sources defined for that character. Each representative glyph is accompanied with its detailed source information provided in alphanumeric form. Altogether, there are nine IRG sources, as shown in *Table 17-1*. Data for these IRG sources are also documented in Unicode Standard Annex #38, “Unicode Han Database (Unihan)”.

Table 17-1. IRG Sources

Name	Source Identity
G source	China PRC and Singapore
H source	Hong Kong SAR
J source	Japan
KP source	North Korea

Table 17-1. IRG Sources (Continued)

Name	Source Identity
K source	South Korea
M source	Macau SAR
T source	Taiwan
U source	Unicode/USA
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 Web site.

Chart for the Main CJK Block. For the CJK Unified Ideographs block (U+4E00..U+9FFF) the glyphs are arranged in the following order: G source and T sources are grouped under the header “C,” and J, K, V and H sources are listed under their respective headers. Each row contains positions for all six sources, and if a particular source is undefined for CJK Unified Ideograph, that position is left blank in the row. This format is illustrated by Figure 17-1. If a character also has a U source, an additional line is used for that character. Note that this block does not contain any characters with M sources. The KP sources are not shown due to lack of reliable glyph information.

Figure 17-1. CJK Chart Format for the Main CJK Block

HEX	C	J	K	V	H
4F1A 人 9.4					
	G0-3B61	T3-2275	J0-3271	K2-216D	V1-4B24
					H-894E

Charts for CJK Extensions. The code charts for all of the extension blocks for CJK Unified Ideographs use a more condensed format. That format dispenses with the “C, J, K, V, and H” 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, and U. The first letters of the source information provide the source type for all sources except G. (For Unicode 6.0, KP sources are omitted from the code charts because of the lack of an appropriately vetted font for display.)

The multicolumn code chart for the CJK Unified Ideographs Extension A block (U+3400..U+4DBF) uses 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 it is associated with more than three sources, as illustrated in Figure 17-2.

Figure 17-2. CJK Chart Format for CJK Extension A

41C9 立 117.5			41DB 竹 118.4				41EE 竹 118.6			
	GHZ-42707.25	T3-3322		GKX-0879.12	T3-3329	V2-7F4B		G5-6334	T4-3975	JA-254D
41CA 立 117.5										
	JA-2549	H-8E85		H-8EFE				V2-7F50		

The multicolumn code charts for the other extension blocks for CJK Unicode Ideographs 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 it is associated with more than two sources, as illustrated in Figure 17-3.

Figure 17-3. CJK Chart Format for CJK Extension B

2000B — 1.3 GHZ-10012.05 T3-2144 ㄷ J3-2E22	2001E — 1.5 GHZ-80007.03 2001F — 1.5 GHZ-80007.04 ㄹ ㄷ	20031 — 1.7 GHZ-10025.01 20032 — 1.7 V0-305F ㅈ ㅊ	20045 — 1.10 GHZ-80007.06 20046 — 1.11 TF-3932 H-9376 ㅍ ㅑ
---	--	---	---

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. However, several additional notational elements described in *Section 17.1, 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. Each CJK unified ideograph in these charts has an annotation identifying it as such. 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-.

Figure 17-4. CJK Chart Format for Compatibility Ideographs

FA15 ⌘ 15.14 J3-775A ≡ 51DE 灑	灑 灑 UTC-00919	FA24 ⌘ 162.4 J4-796E V2-8544 UTC-00851 • a CJK unified ideograph	返 返 返
--	------------------	---	-------

17.3 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 used throughout this core specification.

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.

A range of Unicode code points is expressed as *U+xxxx–U+yyyy* or *xxxx.yyyy*, where *xxxx* and *yyyy* are the first and last Unicode values in the range, and the long 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 100000..10FFFF.

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 17.1, Character Names List*.

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.

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.

Rendering

A figure such as *Figure A-1* depicts how a sequence of characters is typically rendered.

Figure A-1. Example of Rendering

$$\begin{array}{c} \text{A} + \text{ö} \rightarrow \text{Ä} \\ \text{0041} \quad \text{0308} \end{array}$$

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.

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.

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 Web site, <http://www2.arts.gla.ac.uk/IPA/ipa.html>.)

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.

Extended BNF

The Unicode Standard and technical reports use an extended BNF format for describing syntax. As different conventions are used for BNF, *Table A-1* lists the notation used here.

Table A-1. Extended BNF

Symbols	Meaning
$x := \dots$	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
$x \mid y$	either x or y
(x)	for grouping
$x \mid\mid y$	equivalent to $(x \mid y \mid (x y))$
$\{ x \}$	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
$\backslash u1234$	Unicode code points within string literals or character classes
$\backslash U00101234$	Unicode code points within string literals or character classes
$U+HHHH$	Unicode character literal: equivalent to $\backslash uHHHH$
$U-HHHHHHHH$	Unicode character literal: equivalent to $\backslash UHHHHHHHHH$
$[gc=Lu]$	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.

Character Classes. A *code point class* is a specification of an unordered set of code points. Whenever the code points are all assigned characters, it can also be referred to as a *character class*. The specification consists of any of the following:

- A literal code point
- A range of literal code points
- A set of code points having a given Unicode character property value, as defined in the Unicode Character Database (see *PropertyAliases.txt* and *PropertyValueAliases.txt*)
- Non-Boolean properties given as an expression $\langle \text{property} \rangle = \langle \text{property_value} \rangle$ or $\langle \text{property} \rangle \neq \langle \text{property_value} \rangle$, such as “General_Category=Titlecase_Letter”

- Boolean properties given as an expression `<property> = true` or `<property> ≠ true`, such as “`Uppercase=true`”
- Combinations of logical operations on 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:

```

char_class := "[" char_class - char_class "]"           set difference
           := "[" item_list "]"
           := "[" property ("=" | "≠") property_value "]"
item_list  := item (","? item)?
item       := code_point                               either literal or escaped
           := code_point - code_point                 inclusive range

```

Whenever any character could be interpreted as a syntax character, it must be escaped. Where no ambiguity would result (with normal operator precedence), extra square brackets can be discarded. If a space character is used as a literal, it is escaped. Examples are found in *Table A-2*.

Table A-2. Character Class Examples

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
[\u0600-\u06FF - \p{gc=Unassigned}]	all assigned Arabic characters
[\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.”

Operators

Operators used in this standard are listed in *Table A-3*.

Table A-3. Operators

Symbol	Meaning
→	is transformed to, or behaves like
↯	is not transformed to
¬	logical not

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, and the current list of those, with abstracts of their content, is included here for convenient reference.

The Unicode Web site also has many useful online resources. *Section B.6, Other Unicode Online Resources*, provides a guide to the kinds of information available online.

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 Rapporteur 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 Web site.

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 Web site. The Consortium is supported financially solely through membership dues.

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:

<http://www.unicode.org/consortium/utc.html>

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:

<http://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 corresponds to 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 Web site is the source for the most current version of all three categories of technical reports:

<http://www.unicode.org/reports/>

The following sections provide lists of abstracts for current Unicode Technical Standards and Unicode Technical Reports. They are listed numerically within each category. There are gaps in the numerical sequence because some of the reports have been superseded or have been incorporated into the text of the standard.

B.3 Unicode Technical Standards

UTS #6: A Standard Compression Scheme for Unicode

This report presents the specifications of a compression scheme for Unicode and sample implementation.

UTS #10: Unicode Collation Algorithm

This report provides the specification of the Unicode Collation Algorithm, which 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 (DUCET) as the data specifying the default collation order for all Unicode characters.

UTS #18: Unicode Regular Expressions

This document describes guidelines for how to adapt regular expression engines for use with the Unicode Standard.

UTS #22: Character Mapping Markup Language (CharMapML)

This document specifies an XML format for the interchange of mapping data for character encodings. It provides a complete description for such mappings in terms of a defined mapping to and from Unicode code points, and a description of alias tables for the interchange of mapping table names.

UTS #35: Unicode Locale Data Markup Language (LDML)

This document describes an XML format (*vocabulary*) for the exchange of structured locale data. This format is used in the Unicode Common Locale Data Repository.

UTS #37: Unicode Ideographic Variation Database

This document describes the organization of the Ideographic Variation Database and the procedure to add sequences to that database.

UTS #39: Unicode Security Mechanisms

Because Unicode contains such a large number of characters and incorporates the varied writing systems of the world, incorrect usage can expose programs or systems to possible security attacks. This report specifies mechanisms that can be used in detecting possible security problems.

UTS #46: Unicode IDNA Compatibility Processing

Client software, such as browsers and emailers, faces a difficult transition from the version of international domain names approved in 2003 (IDNA2003), to the revision approved in 2010 (IDNA2008). The specification in this document provides a mechanism that minimizes the impact of this transition for client software, allowing client software to access domains that are valid under either system.

B.4 Unicode Technical Reports***UTR #16: UTF-EBCDIC***

This document presents the specifications of UTF-EBCDIC: EBCDIC Friendly Unicode (or UCS) Transformation Format.

UTR #17: Unicode Character Encoding Model

This document clarifies a number of the terms used to describe character encodings, and where the different forms of Unicode fit in. It elaborates the Internet Architecture Board (IAB) three-layer “text stream” definitions into a four-layer structure.

UTR #20: Unicode in XML and Other Markup Languages

This document contains guidelines on the use of the Unicode Standard in conjunction with markup languages such as XML.

UTR #23: The Unicode Character Property Model

This document presents a conceptual model of character properties defined in the Unicode Standard.

UTR #25: Unicode Support for Mathematics

The Unicode Standard includes virtually all standard characters used in mathematics. This set supports a wide variety of math usage on computers, including in document presentation languages like T_EX, in math markup languages like MathML and OpenMath, in internal representations of mathematics for applications like Mathematica, Maple, and MathCAD, in computer programs, and in plain text. This technical report describes the Unicode support for mathematics and gives some of the imputed default math properties for Unicode characters.

UTR #26: Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)

This document specifies an 8-bit Compatibility Encoding Scheme for UTF-16 (CESU) that is intended for internal use within systems processing Unicode to provide an ASCII-compatible 8-bit encoding that is similar to UTF-8 but preserves UTF-16 binary collation. *It is not intended or recommended as an encoding used for open information exchange.* The Unicode Consortium does not encourage the use of CESU-8, but does recognize the existence of data in this encoding and supplies this technical report to clearly define the format and to distinguish it from UTF-8. This encoding does not replace or amend the definition of UTF-8.

UTR #33: Unicode Conformance Model

This report defines conformance terminology, specifies different areas and levels of conformance, and describes what it means to make a claim of conformance or “support” of the standard. This conformance model presented here is not a framework for conformance verification testing.

UTR #36: Unicode Security Considerations

Because Unicode contains such a large number of characters and incorporates the varied writing systems of the world, incorrect usage can expose programs or systems to possible security attacks. This is especially important as more and more products are internationalized. This document describes some of the security considerations that programmers, system analysts, standards developers, and users should take into account, and provides specific recommendations to reduce the risk of problems.

UTR #45: U-Source Ideographs

This document describes U-source ideographs as used by the Ideographic Rapporteur Group (IRG) in its CJK ideograph unification work.

B.5 Unicode Technical Notes

Unicode Technical Notes provide information on a variety of topics related to Unicode and internationalization technologies.

These technical notes are independent publications, not approved by any of the Unicode Technical Committees, nor are they part of the Unicode Standard or any other Unicode specification. Publication does not imply endorsement by the Unicode Consortium in any way. These documents are not subject to the Unicode Patent Policy. Unicode Technical Notes can be found on the Unicode Web site at:

<http://www.unicode.org/notes/>

The technical notes cover the following topics (among others):

- Algorithms
- Collation
- Compression and code set conversions
- Language identification
- Migration of software
- Modern and historical scripts
- Text layout and rendering
- Tutorials
- Social and cultural issues

B.6 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.

Unicode Online Resources

Unicode Web Site

<http://www.unicode.org>

Unicode Anonymous FTP Site

<ftp://ftp.unicode.org>

Charts. The charts section of the Web site provides code charts for all of the Unicode characters, plus specialized charts for normalization, collation, case mapping, script names, and Unified CJK Ideographs.

<http://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.

<http://www.unicode.org/charts/charindex.html>

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.

<http://www.unicode.org/conference/>

E-mail Discussion List. Subscription instructions for the public e-mail discussion list are posted on the Unicode Web site.

FAQ (Frequently Asked Questions). The FAQ pages provide an invaluable resource for understanding the Unicode Standard and its implications for users and implementers.

<http://www.unicode.org/conference/>

Glossary. Online listing of definitions for technical terms used in the Unicode Standard and other publications of the Unicode Consortium.

<http://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.

<http://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.

<http://www.unicode.org/chart/unihan.html>

Policies. These pages describe Unicode Consortium policies on stability, patents, and Unicode Web site privacy. 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.

<http://www.unicode.org/policies/>

Unicode Common Locale Data Repository (CLDR). Machine-readable repository, in XML format, of locale information for use in application and system development.

<http://www.unicode.org/cldr/>

Updates and Errata. This page lists periodic updates with corrections of typographic errors and new clarifications of the text.

<http://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.

<http://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.

<http://www.unicode.org/standard/where/>

How to Contact the Unicode Consortium

The best way to contact the Unicode Consortium to obtain membership information is via the Web site:

<http://www.unicode.org/contacts.html>

The Web site also lists the current telephone, fax, and courier delivery address. The Consortium's postal address is:

P.O. Box 391476
Mountain View, CA 94039-1476
USA

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 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.

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

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.

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 10646-1.

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 10646-1 also contains the contents of Technical Corrigendum No. 2 and all the Editorial Corrigenda to the year 2000.

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 10646 also contains all the Editorial Corrigenda to date.

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 10646.

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.

Unicode 6.1

The Unicode Standard, Version 6.1, is synchronized with the third edition of ISO/IEC 10646.

The synchronization of *The Unicode Standard, Version 6.1*, with ISO/IEC 10646:2012 means that the repertoire, encoding, and names of all characters are identical between the two standard at those version levels. All other changes to the text of 10646 that have a bearing on the text of 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 the term UCS-2 should now be considered obsolete. It no longer refers to an encoding form in either 10646 or the Unicode Standard.

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

<i>Bits</i>	<i>Standard</i>	<i>Binary</i>	<i>Hex</i>	<i>Dec</i>	<i>Char</i>
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

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.

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 the Unicode Standard

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):

- Numbered subset 311 (UNICODE 6.1)
- 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 16.8, *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 Web site. 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 *Section C.5, Identification of Features for the Unicode Standard*.

Appendix D

Changes from Previous Versions

This appendix provides version history of the standard and summarizes updates that have been made to conformance specifications, character content, and data files in the Unicode Character Database since the publication of *The Unicode Standard, Version 5.0*.

D.1 Versions of the Unicode Standard

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 10646-1

Year	Version	Published	ISO/IEC 10646-1
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	Technical Report #4	Matches ISO 10646-1
1996	Unicode 2.0	Addison-Wesley	Matches ISO 10646-1 plus amendments
1998	Unicode 2.1	Technical Report #8	Matches ISO 10646-1 plus amendments
2000	Unicode 3.0	Addison-Wesley	Matches ISO 10646-1 second edition
2001	Unicode 3.1	Standard Annex #27	Matches ISO 10646-1 second edition plus two characters, 10646-2 first edition
2002	Unicode 3.2	Standard Annex #28	Matches ISO 10646-1 second edition plus amendment, 10646-2 first edition
2003	Unicode 4.0	Addison-Wesley	Matches ISO 10646:2003, third version
2005	Unicode 4.1	Web publication	Matches ISO 10646:2003, third version, plus Amd. 1
2006	Unicode 5.0	Addison-Wesley (2007)	Matches ISO 10646:2003, third version, plus Amd. 1, Amd. 2, and four characters from Amd. 3
2008	Unicode 5.1	Web publication	Matches ISO 10646:2003, third version, plus Amd. 1 through Amd. 4
2009	Unicode 5.2	Web publication	Matches ISO 10646:2003, third version, plus Amd. 1 through Amd. 6
2010	Unicode 6.0	Web publication	Matches ISO 10646:2011, second edition
2012	Unicode 6.1	Web publication	Matches ISO 10646:2012, third edition

The Unicode Standard has grown from having 28,294 assigned graphic and format characters in Version 1.0, to having 110,116 characters in Version 6.1. *Table D-2* documents the number of code points allocated in the different versions of the Unicode Standard. The row in *Table D-2* labeled “Graphic + Format” represents the traditional count of Unicode char-

acters and is the typical answer to the question, “How many characters are in the Unicode Standard?” In *Table D-2* the numbers for Han Compatibility include the 12 unified ideographs encoded in the CJK Compatibility Ideographs block.

Table D-2. Allocation of Code Points by Type

	V4.0	V4.1	V5.0	V5.1	V5.2	V6.0	V6.1
Alphabetic, Symbols	13,973	15,117	16,486	18,101	20,588	22,454	23,182
Han (URO)	20,902	20,902	20,902	20,902	20,902	20,902	20,902
Han (URO Extension)		22	22	30	38	38	39
Han Extension A	6,582	6,582	6,582	6,582	6,582	6,582	6,582
Han Extension B	42,711	42,711	42,711	42,711	42,711	42,711	42,711
Han Extension C					4,149	4,149	4,149
Han Extension D						222	222
Han Compatibility	903	1,009	1,009	1,009	1,012	1,012	1,014
Subtotal Han	71,098	71,226	71,226	71,234	75,394	75,616	75,619
Hangul Syllables	11,172	11,172	11,172	11,172	11,172	11,172	11,172
Graphic Characters	96,243	97,515	98,884	100,507	107,154	109,242	109,973
Format Characters	139	140	140	141	142	142	143
Graphic + Format	96,382	97,655	99,024	100,648	107,296	109,384	110,116
Controls	65	65	65	65	65	65	65
Private Use	137,468	137,468	137,468	137,468	137,468	137,468	137,468
Total Assigned	233,915	235,188	236,557	238,181	244,829	246,917	247,649
Surrogate Code Points	2,048	2,048	2,048	2,048	2,048	2,048	2,048
Noncharacters	66	66	66	66	66	66	66
Total Designated	236,029	237,302	238,671	240,295	246,943	249,031	249,763
Reserved Code Points	878,083	876,810	875,441	873,817	867,169	865,081	864,349

Table D-3 lists the allocation of code points by type for earlier, historic versions of the Unicode Standard prior to Version 4.0. In some cases the values in this table differ slightly from summary statistics published in earlier versions of the standard, primarily due to a refined accounting of the allocations in Unicode 1.0.

Table D-3. Allocation of Code Points by Type (Early Versions)

	V1.0.0	V1.0.1	V1.1	V2.0	V2.1	V3.0	V3.1	V3.2
Alphabetic, Symbols	4,734	4,728	6,290	6,491	6,493	10,210	11,798	12,753
Han (URO)		20,902	20,902	20,902	20,902	20,902	20,902	20,902
Han Extension A						6,582	6,582	6,582
Han Extension B							42,711	42,711
Han Compatibility		302	302	302	302	302	844	903
Subtotal Han		21,204	21,204	21,204	21,204	27,786	71,039	71,098
Hangul Syllables	2,350	2,350	6,656	11,172	11,172	11,172	11,172	11,172
Graphic Characters	7,084	28,282	34,150	38,867	38,869	49,168	94,009	95,023
Format Characters	12	12	18	18	18	26	131	133
Graphic + Format	7,096	28,294	34,168	38,885	38,887	49,194	94,140	95,156
Controls	65	65	65	65	65	65	65	65
Private Use	5,632	6,144	6,400	137,468	137,468	137,468	137,468	137,468
Total Assigned	12,793	34,503	40,633	176,418	176,420	186,727	231,673	232,689
Surrogate Code Points				2,048	2,048	2,048	2,048	2,048
Noncharacters	2	2	2	34	34	34	66	66
Total Designated	12,795	34,505	40,635	178,500	178,502	188,809	233,787	234,803
Reserved Code Points	52,741	31,031	24,901	935,612	935,610	925,303	880,325	879,309

D.2 Clause and Definition Updates

Several updates were made to definitions and conformance clauses in Version 5.1 primarily to address potential security exploits. The updates also reflect updated Consortium policies to increase property stability, and include a few other textual clarifications.

Table D-4 provides a list of all clauses and definitions that were clarified, changed, or newly added in Version 5.1.

Table D-4. Version 5.1 Clause and Definition Updates

Number	Clause or Definition	Type of Update
C7	Modification	Clarification
D40	Stable property	Clarification
D51a	Extended base	New
D56a	Extended combining character sequence	New
D60	Grapheme cluster	Changed
D61	Extended grapheme cluster	Changed
D84a	Ill-formed code unit subsequence	New
D85	Well-formed	Changed
D85a	Minimal well-formed code unit subsequence	New
D86	Well-formed UTF-8 code unit sequence	Changed
D121	Case-ignorable	Changed

For Version 5.2, a number of updates were made to incorporate the specification of the normalization algorithm into *Chapter 3, Conformance*, including definitions formerly specified in Unicode Standard Annex #15, “Unicode Normalization Forms.” Other changes include those to tighten security for the handling of noncharacters, and new or changed definitions for deprecated character, code point type, and contributory property. Due to the creation of a new section on normalization, many definitions were renumbered, and a few were moved into other sections.

Table D-5 provides a list of all clauses and definitions that were clarified, changed, newly added, renumbered, or moved in Version 5.2.

Table D-5. Version 5.2 Clause and Definition Updates

Number	Clause or Definition	Type of Update
C7	Modification	Clarification
C13	Normalization	Changed
C14	Normalization	Changed
D10a	Code point type	New
D13	Deprecated character	Clarification
D35a	Contributory property	New
D61a	Dependence	Renumbered (was D102)
D61b	Graphical application	Renumbered (was D103)
D107	Starter	New
D108	Reorderable pair	New
D109	Canonical Ordering Algorithm	New
D110	Singleton decomposition	New
D111	Non-starter decomposition	New
D112	Composition exclusion	New
D113	Full composition exclusion	New
D114	Primary composite	New

Table D-5. Version 5.2 Clause and Definition Updates (Continued)

Number	Clause or Definition	Type of Update
D115	Blocked	New
D116	Non-blocked pair	New
D117	Canonical Composition Algorithm	New
D118	Normalization Form D	New
D119	Normalization Form KD	New
D120	Normalization Form C	New
D121	Normalization Form KC	New
D122 to D133	Hangul syllables	Renumbered (were D107 to D118)
D134	Standard Korean syllable	Renumbered (was D119)
D135 to D138	Case	Renumbered (were D120 to D123)
D139 to D143	Case detection	Renumbered (were D124 to D128)
D144 to D146	Caseless matching	Renumbered (were D129 to D131)

Version 6.0 of the Unicode Standard updated the explanatory text of a few conformance clauses to highlight security considerations. Extensive new text was added to clarify the best practices for using U+FFFD, two new definitions were added in support of the Canonical Composition Algorithm, and new definitions, rules, and explanatory text were added to default case algorithms.

Table D-6 provides a list of all clauses and definitions that were clarified, changed, or newly added in Version 6.0.

Table D-6. Version 6.0 Clause and Definition Updates

Number	Clause or Definition	Type of Update
C7	Modification	Clarification
C10	Character encoding forms	Clarification
D92	UTF-8	Clarification
D93a to D93b	Constraints on Conversion Process	New
D110a to D110b	Canonical Composition Algorithm	New
D111	Canonical Composition Algorithm	Clarification
D138	Default Case Algorithms	Clarification
D144 to D146	Default Case Algorithms	Clarification
D147	Default Case Algorithms	New

Table D-7 provides a list of all clauses and definitions that were clarified, changed, or newly added in Version 6.1.

Table D-7. Version 6.1 Clause and Definition Updates

Number	Clause or Definition	Type of Update
D58	Grapheme base	Clarification
D59	Grapheme extender	Clarification
D60	Grapheme cluster	Clarification
D107	Starter	Clarification
D110	Singleton decomposition	Clarification

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 recombined 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 standardization 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 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 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 *KangXi* 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 12.1, 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 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 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 in Hanoi 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 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 *Table 12-1* in *Section 12.1, Han*. The Extension B collection was first encoded in *The Unicode Standard, Version 3.1*, and is the first collection of unified CJK ideographs to be encoded on Plane 2.

In 2005, the IRG identified a subset of the unified ideographs, called the International Ideograph Core (IICore). This subset is designed to serve as a relatively small collection 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 locales where ideographs are used. The repertoire of the IICore subset is identified with the kIICore key in the *Unihan Database*.

Also in 2005, a small set of ideographs was encoded to support the complete repertoire of the 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 collection of 4,149 characters from various sources. The Extension C collection 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 collection of 222 characters from various sources as documented in *Table 12-1* in *Section 12.1, Han*. The Extension D collection represents a small number of characters which IRG members felt were urgently needed; this collection was first encoded in the Unicode Standard, Version 6.0.

At the present time (early 2012), the IRG is working on a fifth ideographic supplement, Extension E, as well as on an independent set of ideographs, Old Hanzi, for use in representing writing with pre-modern forms. Current IRG work includes submissions from China, Hong Kong, Macao, Taiwan, Japan, South Korea, Vietnam, Malaysia, and the United States.

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 12.1, 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.
- **Pinyin:** The Hanyu Pinyin (Modern Standard Chinese) romanization of the stroke name (as given in the next column), hyphenated to make clear the relationship between the romanization of the stroke name and the acronym value.
- **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. 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	Pinyin	Name	Variant
畫	縮	拼音	筆畫名	異體
一 31D0	H	héng	橫	一
一 4E00, 二 4E8C, 三 4E09, 丁 4E01, 丞 4E1E, 丈 4E08, 世 4E16, 不 4E0D, 土 571F, 上 4E0A, 十 5341, 卅 5345, 七 4E03				
㇇ 31C0	T	tí	提	
㇇ 51AB, 汜 6C3E, 冰 51B0, 治 51B6, 冽 51BD, 冫 6C35, 淋 6DCB, 治 6CBB, 氷 6C3A, 录 5F55, 暴 66B4, 扌 (土 571F), 地 5730, 虫 866B				
丨 31D1	S	shù	豎	/
丨 4E28, 凵 4E29, 上 4E0A, 工 5DE5, 中 4E2D, 串 4E32, 冫 8BA7, 乍 4E4D, 五 4E94, 丑 4E11				
丨 31DA	SG	shù-gōu	豎鉤	
丨 4E85, 水 6C34, 求 6C42, 爭 722D, 事 4E8B				
丿 31D2	P	piě	撇	丿
丿 4E3F, 人 4EBA, 八 516B, 义 4E42, 爻 723B, 禾 79BE, 毛 6BDB, 乏 4E4F, 乖 4E56, 采 91C6, 衣 8863, 行 884C				
丿 31D3	SP	shù-piě	豎撇	丿
乃 4E43, 几 51E0, 人 (人 4EBA), 大 5927, 月 6708, 用 7528, 苐 5C70, 班 73ED, 齊 9F4A				
丶 31D4	D	diǎn	點	丶
丶 4E36, 丸 4E38, 义 4E49, 永 6C38, 冫 51AB, 冰 51B0, 凡 51E1, 丹 4E39, 主 4E3B, 求 6C42, 火 706B, 刃 5203				

Table F-1. CJK Strokes (Continued)




Stroke	Acronym	Pinyin	Name	Variant
畫	縮	拼音	筆畫名	異體
 31CF	N	nà	捺	  
大5927, 人4EBA, 天5929, 入(入5165), 尺5C3A, 走8D70, 是662F, 边8FB9, 廷5EF7				
 31DD	TN	tí-nà	提捺	 
㇇4E40, 𠂇5C10, 八(八516B), 入(入5165), 文(文6587), 攵590A(攵), 廻(廻5EFB)				
 31D5	HZ	héng-zhé	橫折	
冂200CD, 口53E3, 凵56D7, 田7530, 吕5415, 品54C1, 申7533, 甲7532, 圆5706, 𠂇5DEA				
 31C7	HP	héng-piě	橫撇	
又53C8, 双53CC, 𠂇53D2, 今(今4ECA)				
 31D6	HG	héng-gōu	橫鉤	
𠂇4E5B, 𠂇5196, 了4E86, 子5B50, 予4E88, 𠂇77DB, 𠂇758B, 写5199字5B57, 疏758F				
 31D7	SZ	shù-zhé	豎折	
凵200CA, 凵200CB, 山5C71, 牙7259, 互4E92, 乐4E50, 东4E1C, 断65AD, 继7EE7, 彙5F59				
 31D8	SWZ	shù-wān-zuǒ	豎彎左	
凵200CE, 𠂇(𠂇23D92), 肅(肅8085), 嘯562F, 蕭856D, 簫7C2B				

Table F-1. CJK Strokes (Continued)

Stroke	Acronym	Pinyin	Name	Variant
畫	縮	拼音	筆畫名	異體
㇇ 31C4	SW	shù-wān	豎彎	
匚(匚 5338), 區(區 5340), 亡(亡 4EA1), 妄(妄 5984), 四 56DB				
㇏ 31D9	ST	shù-tí	豎提	
冫 2010C, 民 6C11, 艮 826E, 良 826F, 食 98DE, 很 5F88, 狠 72E0, 鄉 9109				
㇇ 31DC	PZ	piě-zhé	撇折	㇇
厶 53B6, 么 5E7A, 糸 7CF8, 絲 7D72, 弘 5F18, 公 516C, 翁 7FC1				
㇏ 31DB	PD	piě-diǎn	撇點	
𠃉 21FE8, 𠃊 5DDDB, 女 5973, 巡 5DE1, 災 707D, 畎 753E, 𧈧 5DE4, 獵 7375				
㇏ 31E2	PG	piě-gōu	撇鉤	
𠃉 4E44, 突 219D1, 𠃊 211A2				
㇏ 31C1	WG	wān-gōu	彎鉤	
𠃉 72AD, 豕 8C55, 豸 8C78, 狐 72D0, 嶽 5DBD, 貓 8C93, 家 5BB6, 逐 9010				
㇏ 31C2	XG	xié-gōu	斜鉤	
弋 5F0B, 戈 6208, 我 6211, 戰 6230				
㇏ 31C3	BXG	biǎn-xié-gōu	扁斜鉤	㇏ ㇏
心 5FC3(楷“心” HKSCS)				

Table F-1. CJK Strokes (Continued)

Stroke	Acronym	Pīnyīn	Name	Variant
畫	縮	拼音	筆畫名	異體
㇇ 31C5	HZZ	héng-zhé-zhé	橫折折	
𠃉534D, 𠃊2067F				
㇈ 31CD	HZW	héng-zhé-wǎn	橫折彎	
爻6BB3, 朶6735, 投6295				
㇉ 31CA	HZT	héng-zhé-tí	橫折提	
讠8BA0, 讠8BA1, 鳩9CE9				
㇊ 31C6	HZG	héng-zhé-gōu	橫折鉤	㇊
㇊200CC, 习4E60, 羽7FBD, 包5305, 匀52FB, 葡8461, 用7528, 青9752, 甫752B, 勺52FA, 月6708, 乚4E5C, 也4E5F				
㇋ 31C8	HZWG	héng-zhé-wǎn-gōu	橫折彎鉤	㇋
九4E5D, 几51E0, 飞98DE, 风98CE, 气6C14, 虱8671, 瘋760B				
㇌ 31DE	SZZ	shù-zhé-zhé	豎折折	㇌
㇌200D1, 𠃍5350, 亞4E9E, 鼎9F0E, 吳5433, 专4E13, 設279AE, 𠃎244F7, 𠃏249A1				
㇍ 31DF	SWG	shù-wǎn-gōu	豎彎鉤	
㇍4E5A, 礼793C, 乱4E71, 几513E, 己5DF1, 巳5DF2, 巳5DF3, 心5FC3, 必5FC5				
㇎ 31CE	HZZZ	héng-zhé-zhé-zhé	橫折折折	
𠃐51F8, 𠃑21E2D				

Table F-1. CJK Strokes (Continued)

Stroke	Acronym	Pīnyīn	Name	Variant
畫	縮	拼音	筆畫名	異體
㇇ 31CB	HZZP	héng-zhé-zhé- piě	橫折折撇	
及 53CA, 𠃉 5EF4, 建 5EFA				
乙 31E0	HXWG	héng-xié-wān- gōu	橫斜彎鉤	
乙 4E59, 𠃉 6C39, 乞 4E5E				
㇈ 31CC	HPWG	héng-piě-wān- gōu	橫撇彎鉤	
𠃉 961D, 𠃉 961F, 𠃉 90AE				
㇉ 31C9	SZWG	shù-zhé-wān- gōu	豎折彎鉤	
𠃉 4E02, 𠃉 4E8E, 号 53F7, 弓 5F13, 强 5F3A, 𠃉 4E10, 𠃉 9A6C				
㇊ 31E1	HZZZG	héng-zhé-zhé- zhé-gōu	橫折折折 鉤	
𠃉 2010E, 乃 4E43, 孕 5B55, 仍 4ECD				
○ 31E3	Q	quān	圈	○
○ 3007, 𠃉 3514, 𠃉 3AB3, 𠃉 3AC8				

References

Citations are given for the standards and dictionaries that were used as the actual resources for *The Unicode Standard*, primarily for Version 1.0. Where a Draft International Standard (DIS) is known to have progressed to International Standard status, the entry for the DIS has been revised. *A revised or reaffirmed edition, with a different date, may have been published subsequently.* For the current version of a standard, see the catalog issued by the standards organization. The Web site of the International Organization for Standardization (<http://www.iso.org>) includes the *ISO Catalogue* and links to the sites of member organizations. Many of the ISO character set standards were originally developed by ECMA and are also ECMA standards.

In general, American library practice has been followed for the romanization of titles and names written in non-Roman script. Exceptions are when information supplied by an author had to be used because the name or title in the original script was unavailable.

R.1 Source Standards and Specifications

This section identifies the standards and specifications used as sources for the Unicode Standard. The section also includes selected current standards and specifications relevant to the use of coded character sets.

AAT: “About Apple Advanced Typography Fonts.” In *TrueType Reference Manual*, Chapter 6: *Font Files*. Apple Computer, ©1997–2002 (last updated 18 Dec 2002).

<http://developer.apple.com/fonts/TTRefMan/RM06/Chap6AATIntro.html>

ANSI X3.4: American National Standards Institute. *Coded character set—7-bit American national standard code for information interchange*. New York: 1986. (ANSI X3.4-1986).

ANSI X3.32: American National Standards Institute. *American national standard graphic representation of the control characters of American national standard code for information interchange*. New York: 1973. (ANSI X3.32-1973).

ANSI Y10.20: American National Standards Institute. *Mathematic signs and symbols for use in physical sciences and technology*. New York: 1988. (ANSI Y10.20-1975 (R1988)).

ANSI Z39.47: American National Standards Institute. *Extended Latin alphabet coded character set for bibliographic use*. New York: 1985. (ANSI Z39.47-1985).

ANSI Z39.64: American National Standards Institute. *East Asian character code for bibliographic use*. New Brunswick, NJ: Transaction, 1991. (ANSI Z39.64-1989).

ARIB STD-B24: Association of Radio Industries and Businesses. *Data Coding and Transmission Specification for Digital Broadcasting*. Tokyo: 2008.

ASMO 449: Arab Organization for Standardization and Metrology. *Data processing 7-bit coded character set for information interchange*. [s.l.]: 1983. (Arab standard specifications, 449-1982). Authorized English translation.

BCP 47: (See RFC 4646 and RFC 4647.)

CCCI: *Zhongwen Zixun Jiaohuanma (Chinese Character Code for Information Interchange)*. Revised edition. Taipei: Xingzhengyuan Wenhua Jianshe Xiaozu (Executive Yuan Committee for Cultural Construction), 1985.

CNS 11643-1986: *Tongyong hanzi biaozhun jiaohuanma (Han character standard interchange code for general use)*. Taipei: Xingzhengyuan (Executive Yuan), 1986.

CNS 11643-1992: *Zhongwen biaozhun jiaohuanma (Chinese standard interchange code)*. Taipei: 1992.

Obsoletes 1986 edition.

DIN 66006: *Informationsverarbeitung—Darstellung von ALGOL-Symbolen auf 5-Spur-Lochstreifen und 80spaltigen Lochkarten. (Information processing—representation of ALGOL symbols on 5-track punched tape and on 80-column punched cards)*. Berlin: Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA), 1965.

Also cited as: *Darstellung von ALGOL/ALCOR-Programmen auf Lochstreifen und Lochkarten*.

EACC: (See ANSI Z39.64.)

ECMA Registry: (See ISO Register.)

ELOT 1373: Hellenic Organization for Standardization (ELOT). *The Greek Byzantine musical notation system*. Athens: 1997.

GB 2312: *Xinxi jiaohuanyong hanzi bianmaj, jibenji (Code of Chinese graphic character set for information interchange, primary set)*. Beijing: Jishu Biaozhun Chubanshe (Technical Standards Press), 1981. (GB 2312-1980).

GB 12345: *Xinxi jiaohuanyong hanzi bianmaj, fuzhuji (Code of Chinese ideogram set for information interchange, supplementary set)*. Beijing: Jishu Biaozhun Chubanshe (Technical Standards Press), 1990. (GB 12345-1990).

GB 13000: *Xinxi jishu—Tongyong duobawei bianma zifuji (UCS)—Diyi bufen: Tixi jiegou yu jiben duowenzhong pingmian (Information technology—Universal multiple-octet coded character set (UCS)—Part 1: Architecture and basic multilingual plane)*. Beijing: Jishu Biaozhun Chubanshe (Technical Standards Press), 1993. (GB 13000.1-93). (ISO/IEC 10646.1-1993).

GB 13134: *Xinxi jiaohuanyong yiwen bianma zifuji (Yi coded character set for information interchange)*, [prepared by] Sichuansheng Minzushiwu Weiyuanhui. Beijing: Jishu Biaozhun Chubanshe (Technical Standards Press), 1991. (GB 13134-1991).

GB 18030: *Xinxi jishu—Xinxi jiaohuan yong hanzi bianma zifuji—Jibenji de kuochong. (Information technology—Chinese ideograms coded character set for information interchange—Extension for the basic set)*. Beijing: Guojiao zhiliang jishu jianduju, 2000. (GB 18030-2000).

GBK: *Xinxi jiaohuanyong hanzi bianma kuozhan guifan (Extended Code of Chinese graphic character set for information interchange)*. Beijing: Zhongguo dianzi gongyebu [and] Guojiao jishu jianduju, 1995.

The Chinese-specific subset of GB 13000.1-93.

GOST 10859-64: USSR. State Committee on Standards, Measures and Measuring Devices of the USSR. *Computational machinery. Alphanumeric Codes for Punchcards and Punctapes*. Moscow: Standards Publishing, 1964.

HKSCS-2001: *Hong Kong Supplementary Character Set – 2001*. Hong Kong: Information Technology Services Department & Official Languages Agency, Government of the Hong Kong Special Administrative Region, 2001.

English: http://www.info.gov.hk/digital21/eng/hkscs/download/e_hkscs.pdf

Chinese: http://www.info.gov.hk/digital21/chi/hkscs/download/c_hkscs.pdf

Irish Standard 434:1999. *Information technology—8-bit single-byte graphic coded character set for Ogham / Teicneolaíocht eolais—Tacar carachtar grafach Oghaim códaithe go haonbheartach le 8 ngiotán.*

ISCII-88: India. Department of Electronics. *Indian script code for information interchange*. New Delhi: 1988.

ISCII-91: India. Bureau of Indian Standards. *Indian script code for information interchange*. New Delhi: 1991.

ISIRI 3342: Institute of Standards and Industrial Research of Iran. *estaandaard-e tabaadol-e ettela'at-e 8 biti-e faarsi = Farsi 8-bit coded character set for information interchange*. Tehran: 1993 (1372 AP). (ISIRI 3342:1993).

ISO Register: International Organization for Standardization. *ISO international register of coded character sets to be used with escape sequences*.

Current register: <http://www.itscj.ipsj.or.jp/ISO-IR/>

ISO 639: International Organization for Standardization. *Code for the representation of names of languages*. [Geneva]: 1988. (ISO 639:1988).

ISO/IEC 646: International Organization for Standardization. *Information technology—ISO 7-bit coded character set for information interchange*. [Geneva]: 1991. (ISO/IEC 646:1991).

ISO/IEC 2022: International Organization for Standardization. *Information processing—ISO 7-bit and 8-bit coded character sets—Code extension techniques*. 3rd ed. [Geneva]: 1986. (ISO 2022:1994).

Edition 4 (ISO/IEC 2022:1994) has title: *Information technology—Character code structure and extension techniques*.

ISO 2033: International Organization for Standardization. *Information processing—Coding of machine-readable characters (MICR and OCR)*. 2nd ed. [Geneva]: 1983. (ISO 2033:1983).

ISO 2047: International Organization for Standardization. *Information processing—Graphical representations for the control characters of the 7-bit coded character set*. [Geneva]: 1975. (ISO 2047:1975).

ISO/IEC 2375: International Organization for Standardization. *Information technology—Procedure for registration of escape sequences and coded character sets*. [Geneva]: 2003. (ISO/IEC 2375:2003).

ISO 3166: International Organization for Standardization. *Codes for the representation of names of countries and their subdivisions*. [Geneva]. Part 1: *Country Codes* (ISO 3166-1:1997). Part 2: *Country subdivision code* (ISO 3166-2:1998). Part 3: *Code for formerly used names of countries* (ISO 3166-3:1999).

ISO/IEC 4873: International Organization for Standardization. *Information technology—ISO 8-bit code for information interchange—Structure and rules for implementation*. [Geneva]: 1991. (ISO/IEC 4873:1991).

ISO 5426: International Organization for Standardization. *Extension of the Latin alphabet coded character set for bibliographic information interchange*. 2nd ed. [Geneva]: 1983. (ISO 5426:1983).

ISO 5426-2: International Organization for Standardization. *Information and documentation—Extension of the Latin alphabet coded character set for bibliographic information interchange—Part 2: Latin characters used in minor European languages and obsolete typography*. [Geneva]: 1996. (ISO 5426-2:1986).

ISO 5427: International Organization for Standardization. *Extension of the Cyrillic alphabet coded character set for bibliographic information interchange*. [Geneva]: 1984. (ISO 5427:1984).

ISO 5428: International Organization for Standardization. *Greek alphabet coded character set for bibliographic information interchange*. [Geneva]: 1984. (ISO 5428-1984).

ISO/IEC 6429: International Organization for Standardization. *Information technology—Control functions for coded character sets*. 3rd ed. [Geneva]: 1992. (ISO/IEC 6429:1992).

ISO 6438: International Organization for Standardization. *Documentation—African coded character set for bibliographic information interchange*. [Geneva]: 1983. (ISO 6438:1983).

ISO 6861:1996. International Organization for Standardization. *Information and documentation—Glagolitic alphabet coded character set for bibliographic information interchange*. [Geneva]: 1996. (ISO 6861:1996).

ISO 6862: International Organization for Standardization. *Information and documentation—Mathematics character set for bibliographic information interchange*. [Geneva]: 1996. (ISO 6862:1996).

ISO/IEC 6937: International Organization for Standardization. *Information processing—Coded character sets for text communication*. [Geneva]: 1984.

Edition 3 (ISO/IEC 6937:2001) has the following title: *Information technology—Coded graphic character set for text communication—Latin alphabet*.

ISO/IEC 8859: International Organization for Standardization. *Information processing—8-bit single-byte coded graphic character sets*. [Geneva]: 1987–.

These parts of ISO/IEC 8859 predate the Unicode Standard, Version 1.0, and were used as resources: Part 1, *Latin alphabet No. 1*; Part 2, *Latin alphabet No. 2*; Part 3, *Latin alphabet No. 3*; Part 4, *Latin alphabet No. 4*; Part 5, *Latin/Cyrillic alphabet*; Part 6, *Latin/Arabic alphabet*; Part 7, *Latin/Greek alphabet*; Part 8, *Latin/Hebrew alphabet*; and Part 9, *Latin alphabet No. 5*.

The other parts of ISO/IEC 8859 are Part 10, *Latin alphabet No. 6*; Part 11, *Latin/Thai alphabet*; Part 13, *Latin alphabet No. 7*; Part 14, *Latin alphabet No. 8 (Celtic)*; Part 15, *Latin alphabet No. 9*; and Part 16, *Latin alphabet No. 10*. There is no Part 12.

ISO 8879: International Organization for Standardization. *Information processing—Text and office systems—Standard generalized markup language (SGML)*. [Geneva]: 1986. (ISO 8879:1986).

ISO 8957: International Organization for Standardization. *Information and documentation—Hebrew alphabet coded character sets for bibliographic information interchange*. [Geneva]: 1996. (ISO 8957:1996).

ISO 9036: International Organization for Standardization. *Information processing—Arabic 7-bit coded character set for information interchange*. [Geneva]: 1987. (ISO 9036:1987).

ISO/IEC 9573-13: International Organization for Standardization. *Information technology—SGML support facilities—Techniques for using SGML—Part 13: Public entity sets for mathematics and science*. [Geneva]: 1991. (ISO/IEC TR 9573-13:1991).

ISO/IEC 9995-7: International Organization for Standardization. *Information technology—Keyboard layouts for text and office systems—Part 7: Symbols used to represent functions*. [Geneva]: 1994. (ISO/IEC 9995-7:1994).

ISO/IEC 10367: International Organization for Standardization. *Information technology—Standardized coded graphic character sets for use in 8-bit codes*. [Geneva]: 1991. (ISO/IEC 10367:1991).

ISO 10585: International Organization for Standardization. *Information and documentation—Armenian alphabet coded character set for bibliographic information interchange*. [Geneva]: 1996. (ISO 10585:1996).

ISO 10586: International Organization for Standardization. *Information and documentation—Georgian alphabet coded character set for bibliographic information interchange*. [Geneva]: 1996. (ISO 10586:1996).

ISO/IEC 10646: International Organization for Standardization. *Information Technology—Universal Multiple-Octet Coded Character Set (UCS)*. [Geneva]: 2003. (ISO/IEC 10646:2003).

ISO/IEC 10646/Amd 1:2005: International Organization for Standardization. *Information Technology—Universal Multiple-Octet Coded Character Set (UCS). Glagolitic, Coptic, Georgian and other characters*. [Geneva]: 2005. (ISO/IEC 10646:2003/Amd 1:2005).

ISO/IEC 10646/Amd 2:2006: International Organization for Standardization. *Information Technology—Universal Multiple-Octet Coded Character Set (UCS). N’Ko, Phags-pa, Phoenician and other characters*. [Geneva]: 2006. (ISO/IEC 10646:2003/Amd 2:2006).

ISO 10754: International Organization for Standardization. *Information and documentation—Extension of the Cyrillic alphabet coded character set for non-Slavic languages for bibliographic information interchange*. [Geneva]: 1996. (ISO 10754:1996).

ISO/TR 11548-1: International Organization for Standardization. *Communication aids for blind persons—Identifiers, names and assignation to coded character sets for 8-dot Braille characters—Part 1: General guidelines for Braille identifiers and shift marks*. [Geneva]: 2001. (ISO/TR 11548-2001).

ISO/TR 11548-2: International Organization for Standardization. *Communication aids for blind persons—Identifiers, names and assignation to coded character sets for 8-dot Braille characters—Part 2: Latin alphabet based character sets*. [Geneva]: 2001. (ISO/TR 11548-2:2001).

ISO 11822: International Organization for Standardization. *Information and documentation—Extension of the Arabic coded character set for bibliographic information interchange*. [Geneva]: 1996. (ISO 11822:1996).

ISO/IEC 14651: International Organization for Standardization. *Information technology—International string ordering and comparison—Method for comparing character strings and description of the common template tailorable ordering*. [Geneva]: 2001. (ISO/IEC 14651:2001).

ISO 15919: International Organization for Standardization. *Information and documentation—Transliteration of Devanagari and related Indic scripts into Latin characters*. [Geneva]: 2001. (ISO 15919:2001).

ISO 15924: International Organization for Standardization. *Information and Documentation—Codes for the representation of names of scripts = Information et documentation—Codes pour la représentation des noms d’écritures*. Bilingual edition = Édition bilingue. [Geneva: 2004]. (ISO 15924:2004).

ISO/IEC TR 19769: International Organization for Standardization. *Information technology—Programming languages, their environments and system software interfaces—Extensions for the programming language C to support new character data types*. [Geneva]: 2004. (ISO/IEC TR 19769:2004).

JIS X 0208: Japanese Industrial Standards Committee. *7 bitto oyobi 8 bitto no 2 baito jouhou koukan you fugouka kanji shuugou (7-bit and 8-bit double byte coded kanji sets for information interchange)*. Tokyo: Japanese Standards Association, 1997. (JIS X 0208:1997).

Revision of the 1990 edition, which was the original source for the Unicode Standard.

JIS X 0212: Japanese Industrial Standards Committee. *Jouhou koukan you kanji fugou—hojo kanji (Code of the supplementary Japanese graphic character set for information interchange)*. Tokyo: Japanese Standards Association, 1990. (JIS X 0212:1990).

JIS X 0213: Japanese Industrial Standards Committee. *7 bitto oyobi 8 bitto no 2 baito jouhou koukan you fugouka kakuchou kanji shuugou (7-bit and 8-bit double byte coded extended kanji sets for information interchange)*. Tokyo: Japanese Standards Association, 2000. (JIS X 0213:2000).

JIS X 0221: Japanese Industrial Standards Committee. *Information Technology—Universal Multiple-Octet Coded Character Set (UCS)—Part 1: Architecture and Basic Multilingual Plane*. Tokyo: Japanese Standards Association, 2001. (JIS X 0221-1:2001).

Identical to ISO/IEC 10646-1:2000.

JIS X 4051-1995: *Line Composition Rules for Japanese Documents*. Japanese Standards Association, 1995.

JIS X 4051:2004: Japanese Industrial Standards Committee. *Nihongo Bunsho no Kumihan Houhou. (Formatting rules for Japanese documents)*. Tokyo: Japanese Standards Association, 2004. (JIS X 4051:2004).

Revision of the 1995 edition, used in Unicode Standard Annex #14, *Line Breaking Properties*.

JIS X 4052:2000: Japanese Industrial Standards Committee. *Nihongo Bunsho no Kumihan Shitei Koukan Keishiki. (Exchange format for Japanese documents with composition markup)*. Tokyo: Japanese Standards Association, 2000. (JIS X 4052:2000).

KPS 9566-97: Committee for Standardization of the Democratic People’s Republic of Korea. *(Code of the Korean graphic character set for information interchange)*. Pyongyang: 1997. (KPS 9566-97).

KPS 10721-2000: Committee for Standardization of the Democratic People’s Republic of Korea. *(Code of the supplementary Korean hanja set for information interchange)*. Pyongyang: 2000. (KPS 10721-2000).

KS C 5601: Korea Industrial Standards Association. *Chongbo kyohwanyong puho (Han’gul mit Hancha)*. Seoul: 1989. (KS C 5601-1987).

KS X 1001: Korean Agency for Technology and Standards. *Chongbo kyohwanyong puho (Han’gul mit Hancha). (Code for information interchange (Hanguel and Hanja))*. Seoul: 1992. (KS X 1001-1992).

Last confirmed 1998. Originally designated as KS C 5601-1992.

KS X 1002: Korean Agency for Technology and Standards. *Chongbo kyohwanyong puho hwakchang se’u. (Extension code sets for information interchange.)* Seoul: 1991. (KS X 1002-1991).

Last confirmed 1996. Originally designated as KS C 5657-1991.

KS X 1026-1:2007 Korean Agency for Technology and Standards. *Information Technology – Universal Multiple Octet Coded Character Set – Hangul, Part 1, Hangul processing guide for information interchange*. 2008.

MIME: (See RFCs 2045-2049, 4648-4649.)

OpenType: *OpenType™ Specification*, version 1.4.

<http://partners.adobe.com/asn/developer/opentype/main.html> (Adobe Systems, ©2000–2003)

<http://www.microsoft.com/typography/otspec/> (Microsoft, ©2001)

The RFCs listed below are available through the *Request for Comments* page of the IETF Web site (<http://www.ietf.org/rfc.html>). This page provides for retrieval by RFC number and includes a list of all RFCs in numerical order (*RFC Index*).

RFC 2045: *Multipurpose Internet Mail Extensions (MIME). Part One: Format of Internet message bodies*, by N. Freed and N. Borenstein. November 1996. (Status: DRAFT STANDARD).

Updated by RFC 2184, RFC 2231.

RFC 2046: *Multipurpose Internet Mail Extensions (MIME). Part Two: Media types*, by N. Freed and N. Borenstein. November 1996. (Status: DRAFT STANDARD).

Updated by RFC 2646, RFC 3798.

RFC 2047: *MIME (Multipurpose Internet Mail Extensions). Part Three: Message header extensions for non-ASCII text*, by K. Moore. November 1996. (Status: DRAFT STANDARD).

Updated by RFC 2184, RFC 2231.

RFC 2048: Obsoleted by RFC 4288, RFC 4289.

RFC 2049: *Multipurpose Internet Mail Extensions (MIME). Part Five: Conformance criteria and examples*, by N. Freed and N. Borenstein. November 1996. (Status: DRAFT STANDARD).

RFC 2152: *UTF-7: A mail-safe transformation format of Unicode*, by D. Goldsmith and M. Davis. May 1997. (Status: INFORMATIONAL).

RFC 3066: Obsoleted by RFC 4646, RFC 4647.

RFC 3629: *UTF-8: A transformation format of ISO 10646*, by F. Yergeau. November 2003. (Also STD0063). (Status: STANDARD).

RFC 4288: *Media type specifications and registration procedures*, by N. Freed and J. Klensin. December 2005. (Also BCP0013). (Status: BEST CURRENT PRACTICE).

RFC 4289: *Multipurpose Internet Mail Extensions (MIME). Part Four: Registration procedures*, by N. Freed and J. Klensin. December 2005. (Also BCP0013). (Status: BEST CURRENT PRACTICE).

RFC 4646: *Tags for identifying languages*, edited by A. Phillips and M. Davis. 2006. (Also BCP0047). (Status: BEST CURRENT PRACTICE).

RFC 4647: *Matching of language tags*, edited by A. Phillips and M. Davis. 2006. (Also BCP0047). (Status: BEST CURRENT PRACTICE).

SI 1311.1: Standards Institution of Israel. *Information technology: ISO 8-bit coded character set with Hebrew points*. [Tel Aviv: 1996]. (SI 1311.1 (1996)).

SI 1311.2: Standards Institution of Israel. *Information technology: ISO 8-bit coded character set with Hebrew accents*. [Tel Aviv: 1996]. (SI 1311.2 (1996)).

SLS 1134: Sri Lanka Standards Institution. *Sinhala character code for information interchange*. Colombo: 1996. (SLS 1134: 1996).

TIS 620-2529: Thai Industrial Standards Institute, Ministry of Industry. *Thai Industrial Standard for Thai character code for computer*. Bangkok: 1986. (TIS 620-2529–1986).

TIS 620-2533: Thai Industrial Standards Institute. *Standard for Thai character codes for computers*. Bangkok: 1990. (TIS 620-2533–1990). ISBN 974-606-153-4.

In Thai. Online version: <http://www.nectec.or.th/it-standards/std620/std620.htm>

Extensible Markup Language (XML) 1.0. 4th ed. (W3C Recommendation 16 August 2006). Editors: Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, [and] François Yergeau.

<http://www.w3.org/TR/2006/REC-xml-20060816/>

Extensible Markup Language (XML), 1.1. 2nd ed. (W3C Recommendation 16 August 2006). Editors: Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, [and] John Cowan.

<http://www.w3.org/TR/2006/REC-xml11-20060816/>

R.2 Source Dictionaries for Han Unification

Dae Jaweon. Seoul: Samseong Publishing Co. Ltd., 1988.

Dai Kan-Wa Jiten / Morohashi Tetsuji cho. Shu teiban. Tokyo: Taishukan Shoten, Showa 59-61, [1984–86].

Hanyu Da Zidian. 1st ed. Chendu: Sichuan Cishu Publishing, 1986.

KangXi Zidian. 7th ed. Beijing: Zhonghua Bookstore, 1989.

R.3 Other Sources for the Unicode Standard

General

ALA-LC Romanization Tables: Transliteration Schemes for Non-Roman Scripts, Approved by the Library of Congress and the American Library Association. Tables compiled and edited by Randall K. Barry. Washington, DC: Library of Congress, 1997. ISBN 0-8444-0940-5.

Alchemical Symbols

Berthelot, Marcelin. *Collection des anciens alchimistes grecs*. 3 vols. Paris: G. Steinheil, 1888.

Berthelot, Marcelin. *La chimie au moyen âge*. 3 vols. Osnabrück: O. Zeller, 1967.

Lüdy-Tenger, Fritz. *Alchemistische und chemische Zeichen*. Würzburg: JAL-reprint, 1973.

Schneider, Wolfgang. *Lexicon alchemistisch-pharmazeutischer Symbole*. Weinheim/Bergstr.: Verlag Chemie, 1962.

Avestan

Geldner, Karl F. *Avesta: The Sacred Books of the Parsis*. Stuttgart: W. Kohlhammer, 1880.

Reprinted, with an introduction in Persian by Dr. Jaleh Amouzgar. Tehran: Asatir, 2003. ISBN 964-331-126-0.

Hoffmann, Karl, and B. Forssman. *Avestische Laut- und Flexionslehre*. Innsbruck: Innsbrucker Beiträge zur Sprachwissenschaft, 1996. ISBN 3851246527.

Oryan, Said. *Pahlavi-Pazand Glossary: Farhang ī Pahlavi*. Tehran: Research Institute for Islamic Culture and Art, 1999 (1377 AP). (Language and Literature, 4). ISBN 964-471-414-8.

Reichelt, Hans. *Avesta Reader: An Approach to the Zoroaster's Gathas and New Avestan Texts*. Translated and annotated with Persian translation of hymns and texts by Jalil Doostkhah. Tehran: Qoqnoos Publishing, 2004 (1383 AP). ISBN 964-311-473-2.

Balinese

Medra, Nengah. *Pedoman Pasang Aksara Bali*. Denpasar: Dinas Kebudayaan Propinsi Bali, 2003.

Menaka, Made. *Kamus Kawi Bali / olih, made Menaka*. Singaraja: Yayasan Kawi Sastra Mandala, 1990.

Simpem, I Wayan. *Pasang Aksara Bali*. Denpasar: Upada Sastra, 1992.
Also published: Denpasar: Dinas Pengajaran Daerah Tingkat I Bali, 1979.

Bamum

Dugast, J., and M. D. W. Jeffreys. *L'écriture des bamum: sa naissance, son évolution, sa valeur phonétique, son utilisation*. Mémoires de l'Institut Français d'Afrique Noire, Centre du Cameroun, 1950.

Nchare, Oumarou. *The Writing of King Njoya: Genesis, Evolution, Use*. Foumban: Palais des Rois Bamoun, Maison de la Culture, [s.d.].

Schmitt, Alfred. *Die Bamum-Schrift*. Band I: *Text*. Wiesbaden: Harrassowitz, 1963.

Batak

Kozok, Uli. *Warisan leluhur: sastra lama dan aksara Batak*. Jakarta: École française d'Extrême Orient, 1999. ISBN 979-9023-33-5.

Meerwaldt, J H. *Handleiding tot de beoefening der Bataksche taal*. Leiden: E.J. Brill, 1904.

Tuuk, Herman Neubronner van der. *A Grammar of Toba Batak*. Translated by Jeune Scott-Kemball, edited by Andries Teeuw and R. Roolvink. The Hague: Nijhoff, 1971.

First English edition of Tobasche spraakkunst, 1864-1867.

Brahmi

Baums, Stefan. "Towards a Computer Encoding for Brāhmī." In *Script and Image: Papers on Art and Epigraphy*, edited by Adalbert J. Gail, Gerd J. R. Mevissen and Richard Salomon, vol. 11.1, 111–143. Delhi: Motilal Banarsidass Publishers, 2006.

Bühler, G. "The Bhattiprolu Inscriptions." In *Epigraphia Indica: A Collection of Inscriptions Supplementary to the Corpus Inscriptionum Indicarum of the Archaeological Survey*, vol. 2, 323–329. Calcutta: Epigraphia Indica, 1894.

Dani, Ahmad Hasan. *Indian Palaeography*. 2nd edition. New Delhi: Munshiram Manoharlal Publishers, 1986.

Mahadevan, Iravatham. *Early Tamil Epigraphy: From the Earliest Times to the Sixth Century A.D.* Chennai, India: Cre-A, 2003. (Harvard Oriental Series, vol. 62.)

Canadian Aboriginal Syllabics

Canadian Aboriginal Syllabic Encoding Committee. *Repertoire of Unified Canadian Aboriginal Syllabics Proposed for Inclusion into ISO/IEC 10646: International Standard Universal Multiple-Octet Coded Character Set*. [Canada]: CASEC, [1994].

Carian

Adiego, Ignacio-Javier. *The Carian Language*. Leiden; Boston: Brill, 2007.

Melchert, H. Craig. “Carian.” In *The Cambridge Encyclopedia of the World’s Ancient Languages*, edited by Roger Woodard, 609–613. Cambridge: Cambridge University Press, 2004. ISBN-13: 978-0521562560.

Chakma

Cānmā, Cīrajyoti and Maṅgal Cāṅgmā. *Cāṅmār āg p u d h i* = Chakma primer. Rāṅnamāṭi: Cāṅmābhāṣā Prakāśanā Pariṣad. 1982.

Khisa, Bhagadatta. *Cāṅmā pattham pāt* = Chakma primer. Rāṅnamāṭi: Tribal Cultural Institute, 2001.

Cham

Aymonier, Étienne, and Antoine Cabaton. *Dictionnaire Čam-Français*. Paris, 1906.

Bùi Khánh Thế. *Từ điển Chăm-Việt: Inālang cam-biet đām*. [Hồ Chí Minh]: Nhà xuất bản Khoa Học Xã Hội, 1995.

Kōno Rokurō, Chino Eiichi, and Nishida Tatsuo. *The Sanseido Encyclopaedia of Linguistics. Volume 7: Scripts and Writing Systems of the World* [*Gengogaku dai ziten (bekkan) sekai mozi ziten*]. Tokyo: Sanseido Press, 2001. ISBN 4-385-15177-6.

Cherokee

Alexander, J. T. *A Dictionary of the Cherokee Indian Language*. [Sperry, Oklahoma?]: Published by the author, 1971.

Holmes, Ruth Bradley. *Beginning Cherokee*, by Ruth Bradley Holmes and Betty Sharp Smith. 2nd ed. Norman: University of Oklahoma Press, 1977. ISBN 0-8061-1464-9.

New Echota Letters: Contributions of Samuel A. Worcester to the Cherokee Phoenix, edited by Jack Frederick Kilpatrick and Anna Gritts Kilpatrick. Dallas: Southern Methodist University Press, [s.d.].

Includes reprint of an article by S. A. Worcester, which appeared in the *Cherokee Phoenix*, Feb. 21, 1828.

Coptic

Browne, Gerald M. *Old Nubian Grammar*. München: Lincom Europa, 2002. (*Languages of the world: Materials*, 330). ISBN 3-89586-893-0 (pbk.).

Kasser, Rodolphe. “La ‘Genève 1986’: une nouvelle série de caractères typographiques copptes, protocoptes et vieux-coptes créée à Genève.” *Bulletin de la Société d’égyptologie de Genève*, 12 (1988): 59–60. ISSN 0255-6286.

Kasser, Rodolphe. “A Standard System of Sigla for Referring to the Dialects of Coptic.” *Journal of Coptic Studies*, 1 (1990): 141–151. ISSN 1016-5584.

Cypriot

See Linear B and Cypriot.

Deseret

Encyclopedia of Mormonism, entry for “Deseret Alphabet.”

New York: Macmillan, 1992. ISBN 0-02-904040-X.

Monson, Samuel C. *Representative American Phonetic Alphabets*. New York: 1954.

Ph.D. dissertation—Columbia University.

Egyptian Hieroglyphs

Allen, James P. *Middle Egyptian: An Introduction to the Language and Culture of Hieroglyphs*. Cambridge: Cambridge University Press, 1999. ISBN 0-521-77483-7.

Gardiner, Alan H. *Catalogue of the Egyptian Hieroglyphic Printing Type, from Matrices Owned and Controlled by Dr. Alan H. Gardiner, in Two Sizes, 18 Point, 12 Point with Intermediate Forms*. Oxford: Oxford University Press, 1928.

Gardiner, Alan H. “Additions to the New Hieroglyphic Fount (1928).” *The Journal of Egyptian Archaeology*, 15 (1929): 95. ISSN 0307 5133.

Gardiner, Alan H. “Additions to the New Hieroglyphic Fount (1931).” *The Journal of Egyptian Archaeology*, 17 (1931): 245–247. ISSN 0307 5133.

Gardiner, Alan H. *Supplement to the Catalogue of the Egyptian Hieroglyphic Printing Type, Showing Acquisitions to December 1953*. Oxford: Oxford University Press, 1953.

Gardiner, Alan H. *Egyptian Grammar: Being an Introduction to the Study of Hieroglyphs*. 3rd edition. London: Oxford University Press, 1957. ISBN 0-900416-35-1.

Ethiopic

Armbruster, Carl Hubert. *Initia Amharica: An Introduction to Spoken Amharic*. Cambridge: Cambridge University Press, 1908–1920.

Launhardt, Johannes. *Guide to Learning the Oromo (Galla) Language*. Addis Ababa: Launhardt, [1973?].

Leslau, Wolf. *Amharic Textbook*. Wiesbaden: Harrassowitz; Berkeley: University of California Press, 1968.

Glagolitic

Glagolitica: zum Ursprung der slavischen Schriftkultur, herausgegeben von Heinz Miklas, unter der Mitarbeit von Sylvia Richter und Velizar Sadovski. Wien: Verlag der Österreichischen Akademie der Wissenschaften, 2000. (*Schriften der Balkan-Kommission, Philologische Abteilung*, 41). ISBN 3-7001-2895-9.

Khaburgaev, Georgii Aleksandrovich. *Staroslavianskii iazyk*. Izd. 2-e, perer. i dop. Moskva: Prosveshchenie, 1986.

Žubrinic, Darko. *Hrvatska glagoljica: biti pismen—biti svoj*. Zagreb: Hrvatsko književno društvo sv. Jeronima (sv. Cirila i Metoda): Element, 1996. ISBN 953-6111-35-7.

Gothic

Ebbinghaus, Ernst. “The Gothic Alphabet.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Greek Editorial Marks

Austin, Colin. *Comicorum Graecorum Fragmenta in Papyris Reperta*, ed. Colinus Austin. Berolini [Berlin], Novi Eboraci [New York]: de Gruyter, 1973, p. 29. ISBN 3110024012.

Homer. *Iliad. Homeri Ilias*, edidit Thomas W. Allen. 3 vols. Oxonii [Oxford]: e typographeo Clarendoniano [Clarendon Press], 1931, vol. 2: pp. 39, 234.

The Oxyrhynchus Papyri, Part XV, edited with translations and notes by Bernard P. Grenfell and Arthur S. Hunt. London: Egypt Exploration Society, 1921, p. 56. (*Egypt Exploration Society, Graeco-Roman Memoirs*, 18).

Imperial Aramaic

Driver, G. R. *Semitic Writing from Pictograph to Alphabet*. 3rd ed. by S. A. Hopkins. London: Oxford University Press for the British Academy, 1976. ISBN 9780197259177.

Lidzbarski, Mark. *Handbuch der nordsemitischen Epigraphik nebst ausgewählten Inschriften*. Hildesheim: Georg Olms Verlagsbuchhandlung, 1962.

Reprint of 1898 edition.

Naveh, Joseph. *Early History of the Alphabet: An Introduction to West Semitic Epigraphy and Palaeography*. Jerusalem: Magnes Press, the Hebrew University, 1987. ISBN 965-223-436-2.

Porten, Bezalel, and Ada Yardeni. *Textbook of Aramaic Documents from Ancient Egypt*. 4 vols. Jerusalem: Hebrew University, 1986–1999. ISBN 9652220752 (v. 1), 9653500031 (v. 2), 9653500147 (v. 3), 9653500899 (v. 4).

Rosenthal, Franz. *A Grammar of Biblical Aramaic*. 7th rev. ed. Wiesbaden: Harrassowitz, 2006. ISBN 3-447-05251-1.

Inscriptional Parthian and Inscriptional Pahlavi

Akbarzādeh, Dāriyūš. *Katibe-hā-ye Pahlavi-ye Aškāni (Pārti) = Parthian Inscriptions*. Vol. 2. Tehran: Pazineh Press, 2002 (1381 AP). ISBN 964-5722-74-8.

Akbarzādeh, Dāriyūš. *Katibe-hā-ye Pahlavi: sang-negāre, sekke, mohr, asar-e mohr, zarf-nebešte = Pahlavi Inscriptions: Inscriptions, Coins, Seals, Sealing Impression*. Vol. I. Tehran: Pazineh Press, 2003 (1382 AP). ISBN 964-5722-44-6.

Nyberg, Henrik Samuel. *A Manual of Pahlavi*. 2 vols. Wiesbaden: Harrassowitz, 1964–1974. ISBN 9783447015806 (vol. 2).

Reprinted: Tehran: Asatir, 2003. ISBN 964-331-132-5, 964-331-131-7.

Oryan, Saeed. *Rahnmā-ye katibe-hā-ye Irāni-ye miyāne Pahlavi-Pārti = Manual of Middle Iranian Inscriptions (Parthian-Pahlavi)*. Tehran: Iranian Cultural Heritage Organization, 2003 (1382 AP). ISBN 964-7483-71-6.

Rezai Baghbidi, Hassan. *Dastur-e Zabān-e Pārti (Pahlavi-e Aškāni) = A Grammar of Parthian (Arsacid Pahlavi)*. Iranian Academy of Persian Language and Literature, 2002 (1381 AP). ISBN 964-7531-05-2.

International Phonetic Alphabet

Esling, John. “Computer Coding of the IPA: Supplementary Report.” *Journal of the International Phonetic Association*, 20.1 (1990): 22–26.

International Phonetic Association. *Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge: Cambridge University Press, 1999. ISBN 0-521-65236-7; 0-521-63751-1 (pbk.).

International Phonetic Association.

<http://www2.arts.gla.ac.uk/IPA/ipa.html>

Journal of the International Phonetic Association, 24.2 (1994): 95–98, and 25.1 (1995): 21.

Pullum, Geoffrey K. “Remarks on the 1989 Revision of the International Phonetic Alphabet.” *Journal of the International Phonetic Association*, 20.1 (1990): 33–40.

Pullum, Geoffrey K., and William A. Ladusaw. *Phonetic Symbol Guide*. 2nd ed. Chicago: University of Chicago Press, 1996. ISBN 0-226-68535-7; 0-226-68536-5 (pbk.).

Wells, John Christopher. *Accents of English*. Cambridge, New York: Cambridge University Press, 1982.

Vol. 1: *Introduction*. ISBN 0-521-22919-7; ISBN 0-521-29719-2 (pbk.); vol. 2: *The British Isles*. ISBN 0-521-24224-X, ISBN 0-521-28540-2 (pbk.); vol. 3: *Beyond the British Isles*. ISBN 0-521-24225-8, ISBN 0-521-28541-0 (pbk.).

Javanese

Bohatta, Hanns. *Praktische Grammatik der javanischen Sprache, mit Lesestücken, einem javanisch-deutschen und deutsch-javanischen Wörterbuch*. Wien, Pest, Leipzig: Hartleben, [1892]. (Kunst der Polyglottie, 39).

Rochadi GK, R. H., and R. L. Sadeli Erawan BK. *Cacarakan aksara Sunda*. Bandung: Harisma, 1984.

Roorda, T. *Javaansche grammatica, benevens een leesboek tot oefening in de javaansche taal*. Amsterdam: Johannes Müller, 1855.

Walbeehm, A. H. J. G. *Javaansche spraakkunst: schrift, uitspraak, taalsoorten en woordafleiding*. Leiden: E. J. Brill, 1905.

Kaithi

Bihar High Court of Judicature. *Selection of Hindusthani Documents from the Courts of Bihar*, compiled by S. K. Das. Patna, Bihar: Superintendent, Government Printing, 1939.

Grierson, George A. *A Handbook to the Kaithi Character*. 2nd rev. ed. Calcutta: Thacker, Spink & Co., 1899.

Revised edition of *A Kaithi Handbook*, 1881.

King, Christopher R. *One Language, Two Scripts: The Hindi Movement in Nineteenth Century North India*. Bombay: Oxford University Press, 1994.

Kayah Li

Bennett, J. Fraser. *Kayah Li Script: A Brief Description*. Urbana-Champaign: University of Illinois, 1993.

Karenni Literature Department. *Ka¹ya³lhi¹-Ku³la³ Nghôchozha³: The Modern Western Kayah Li-English Lexicon*. [Chiang Mai]: Payap University, 1994. [without tones = Kayalhi-Kula Nghôchozha]

Solnit, David B. *Eastern Kayah Li: Grammar, Texts, Glossary*. Honolulu: University of Hawai‘i Press, 1997. ISBN 0-8248-1743-5.

Kharoshthi

Glass, Andrew. *A Preliminary Study of Kharoshthi Manuscript Paleography*. 2000.

Thesis (M.A.), University of Washington, 2000.

Glass, Andrew. “Kharoṣṭhī Manuscripts: A Window on Gandhāran Buddhism.” *Nagoya Studies in Indian Culture and Buddhism*, 24 (2004): 129–152. ISSN 0285-7154.

Salomon, Richard. *Ancient Buddhist Scrolls from Gandhāra: The British Library Kharoshthi Fragments*. Seattle: University of Washington Press; London: British Library, 1999. ISBN 029597768X; 0295977698 (pbk).

Lepcha

Mainwaring, G. B. *A Grammar of the Rong (Lepcha) Language as it Exists in the Dorjeling and Sikim Hills*. Delhi: Daya Publishing House, 1985 (1876).

Plaisier, H. “A Brief Introduction to Lepcha Orthography and Literature.” *Bulletin of Tibetology* 41:1 (2005), 7–24.

Plaisier H. *A Grammar of Lepcha*. Leiden: Brill, 2007. (Brill’s Tibetan Studies Library, Languages of the Greater Himalayan Region 5).

Limbu

Cemjonga, Imana Simha, and Bairagi Kaila, eds. *Limbu-Nepali-Angreji śabdakoś*. [Limbu-Nepali-English Dictionary.] Kathmandu: Royal Nepal Academy, 2059 [2002].

Includes an introduction describing the Limbu script.

Cemjonga, Imana Simha. *Yakthun-Pene-Mikphula Pancheka*. = *Limbu-Nepali-Angareji śabdakoś*. = *Limbu-Nepali-English Dictionary*. [Lekhaka] Imanasimha Cemajon. [Kathmandu]: Nepala Ekedemi [2018 vi., i.e., 1962].

In Devanagari script. Author also known as Chemjong, Iman Singh.

Driem, George van. *A Grammar of Limbu*. Berlin, New York: Mouton de Gruyter, 1987. (Mouton grammar library, 4.) ISBN 0-89925-345-8.

Appendix: *Anthology of Kiranti scripts*, pp. 550–558.

Shafer, Robert. *Introduction to Sino-Tibetan*. Wiesbaden: Harrassowitz, 1966–1974.

Published in five parts with continuous pagination.

Sprigg, R. K. “Limbu Books in the Kiranti Script.” In International Congress of Orientalists (24th: 1957: Munich). *Akten des Vierundzwanzigsten Internationalen Orientalisten-Kongresses München 28. August bis 4. September 1957*, hrsg. von Herbert Franke. Wiesbaden: Deutsche Morgenländische Gesellschaft, in Kommission bei Franz Steiner Verlag, 1959.

The modern script described in this work is now outdated.

Sprigg, R. K. [Review of van Driem (1987)]. *Bulletin of the School of Oriental and African Studies, University of London*, 52:1 (1989), 163–165.

Subba, B. B. *Limbu, Nepali, English Dictionary*. Gangtok: Text Book Unit, Directorate of Education, Govt. of Sikkim, 1979 [i.e. 1980].

Cover title: Yakhthun-Pene-Mikphula-panchekva. In Limbu and Devnagari scripts.

Subba, B. B. *Yakhthun huḷsiḅlam* (“Limbu self-teaching method”) = *Limbu akṣar gāiḅ* (“Limbu letter guide”). Gangtok: Kwaliti Stores, 1991?

In Nepali and Limbu.

Yoñhāñ, Khel Rāj. *Limbū Nepālī śabdakoś*. [Lalitpur]: 2052 B.S. [i.e. 1995].

In Limbu script.

Linear B and Cypriot

Bennett, Emmett L. “Aegean Scripts.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Chadwick, John. *The Decipherment of Linear B*. 2nd ed. London: Cambridge University Press, 1967 [i.e. 1968].

Chadwick, John. *Linear B and Related Scripts*. Berkeley: University of California Press; [London]: British Museum, 1987. (*Reading the Past*, v. 1). ISBN 0-520-06019-9.

Hooker, J. T. *Linear B: An Introduction*. Bristol: Bristol Classical Press, 1980. ISBN 0-906515-69-6.

Corrected printing published 1983. ISBN 0-906515-69-6; 0-906515-62-9 (pbk.).

International Colloquium on Mycenaean Studies (3rd: 1961: Racine, WI). *Mycenaean Studies: Proceedings of the Third International Colloquium for Mycenaean Studies held at “Wingspread,” 4–8 September 1961*, edited by Emmett L. Bennett, Jr. Madison: University of Wisconsin Press, 1964.

Appendix: The Wingspread Convention for the Transcription of Mycenaean (Linear B) Texts: pp. 254–262.

Masson, Olivier. *Les Inscriptions chypriotes syllabiques: recueil critique et commenté*. Réimpr. augm. Paris: E. de Boccard, 1983.

Sampson, Geoffrey. *Writing Systems: A Linguistic Introduction*. Stanford, CA: Stanford University Press, 1985. ISBN 0-8047-1254-9.

Also published: London, Hutchinson. ISBN 0-09-156980-X; 0-09-173051-1 (pbk.).

Ventris, Michael. *Documents in Mycenaean Greek*. 1st ed. by Michael Ventris and John Chadwick with a foreword by Alan J. B. Wace. 2nd ed. by John Chadwick. Cambridge: Cambridge University Press, 1973. ISBN 0-521-08558-6.

Lisu

Bya, Yuliya. *Li-su Tho Uh Ba Pa Pha Tso So Du (Lisu Alphabet Primer)*. Chiang Mai: Christian Literature Fellowship, 2000.

Xu, Lin, Yuzhang Mu, and Xingzhi Gai, eds. *Lisuyu Jianzhi (A Sketch of the Lisu Language)*. Beijing: The Ethnic Publishing House, 1986. (*Chinese Minority Language Sketch Series*.)

Yunnan Minority Language Commission, and Weixi Culture and Education Bureau, eds. *Li-su Tho Uh Tso So Du (Lisu Primer)*. Kunming: Yunnan Nationality Publishing House, 1981.

Zhu, Faqing. *Li-su Be Xuh Ngo Bae Khuh Tae Du Ra (Small Lisu-Chinese Dictionary)*. Dehong: Dehong Nationality Publishing House, 1984.

Lycian

Carruba, O. “La scrittura licia.” *Annali della scuola normale superiore di Pisa, classe di letter e filosofia*. 3rd series. 8 (1978):849–867.

Melchert, H. Craig. “Lycian.” In *The Cambridge Encyclopedia of the World’s Ancient Languages*, edited by Roger Woodard, 591–600. Cambridge: Cambridge University Press, 2004. ISBN-13: 978-0521562560.

Lydian

Gérard, Raphaël. *Phonétique et morphologie de la langue lydienne*. Louvain-la-Neuve: Peeters, 2005.

Gusmani, Roberto. *Lydisches Wörterbuch mit grammatischer Skizze und Inschriftensammlung*. Heidelberg: Carl Winter, 1964.

Melchert, H. Craig. “Lydian.” In *The Cambridge Encyclopedia of the World’s Ancient Languages*, edited by Roger Woodard, 601–608. Cambridge: Cambridge University Press, 2004. ISBN-13: 978-0521562560.

Mandaic

Daniels, P. “Aramaic Scripts for Aramaic Languages.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Häberl, C. “Iranian Scripts for Aramaic Languages: The Origin of the Mandaic Script,” *Bulletin of the American Schools of Oriental Research*, No. 341 (Feb., 2006), pp. 53–62.

Coulmas, Florian. *The Blackwell Encyclopedia of Writing Systems*. Oxford, Cambridge: Blackwell, 1999. ISBN 0-631-19446-0, 0-631-21481-X (pbk.).

Mandaean script, p. 320.

Mathematical Symbols

Mathematical Markup Language (MathML) Version 2.0. (W3C Recommendation 21 February 2001). Editors: David Carlisle, Patrick Ion, Robert Miner, [and] Nico Popplier.

Latest version: <http://www.w3.org/TR/MathML2/>

STIPub Consortium. STIX (Scientific and Technical Information Exchange) Project.

<http://www.ams.org/STIX/>

Swanson, Ellen. *Mathematics into Type*. Updated ed. by Arlene O’Sean and Antoinette Schleyer. Providence, RI: American Mathematical Society, 1999. ISBN 0-8218-1961-5.

Meetei Mayek

Chelliah, Shobhana L. *A Grammar of Meithei*. Berlin and New York: Mouton de Gruyter, 1997. ISBN 978-3-11-014321-8.

Debendra Singh, N. *Evolution of Manipuri Script*. [Imphal]: Manipur University, Centre for Manipuri Studies, 1990. (Research Report, 5).

Meroitic

Griffith, F. Ll. *Karanòg: The Meroitic inscriptions of Shablùl and Karanòg*. Philadelphia: University Museum, 1911.

Millet, N. B. “The Meroitic script.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Rilly, Claude. *La langue du royaume de Méroé: un panorama de la plus ancienne culture écrite d’Afrique subsaharienne*. Paris: Librairie Honoré Champion, 2007.

Miao

Enwall, Joakim. *A Myth Become Reality: History and development of the Miao written language*. 2 vols. Stockholm: Institute of Oriental Languages, Stockholm University, 1994–1995. (Stockholm East Asian monographs no. 5-6.)

Xiong Yuyou. *Miao zu wen hua shi = A Cultural History of the Miao Nationality*. Kunming Shi: Yunnan min zu chu ban she, 2003.

Musical Symbols

Catholic Church. *Graduale Sacrosanctae Romanae Ecclesiae de Tempore et de Sanctis SS. D. N. Pii X. Pontificis Maximi*. Parisiis: Desclée, 1961. (*Graduale Romanum*, no. 696).

Gazimihal, Mahmut R. *Anadolu türküleri ve mûsikî istikbâlimiz* [by] Mahmut Ragıp. [Istanbul]: Mârifet Matbaası, 1928.

Heussenstamm, George. *Norton Manual of Music Notation*. New York: W.W. Norton, 1987. ISBN 0-393-95526-5 (pbk.).

Kennedy, Michael. *Oxford Dictionary of Music*. Oxford, New York: Oxford University Press, 1985. ISBN 0-19-311333-3.

2nd ed. published 1994. ISBN 0-19-869162-9.

New Encyclopedia Britannica. 15th ed. Entry for “Music.”

The New Harvard Dictionary of Music, edited by Don Michael Randel. Cambridge, MA: Belknap Press of Harvard University Press, 1986. ISBN 0-674-61525-5.

Ottman, Robert W. *Elementary Harmony: Theory and Practice*. 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1970. ISBN 0-13-257451-9.

Fifth ed. published 1998. ISBN 0-13-281610-5.

Rastall, Richard. *The Notation of Western Music: An Introduction*. London: Dent, 1983. ISBN 0-460-04205-X.

Also published: New York: St. Martin’s Press, 1982. ISBN 0-312-57963-2.

Read, Gardner. *Music Notation: A Manual of Modern Practice*. Boston: Allyn and Bacon, 1964.

Stone, Kurt. *Music Notation in the Twentieth Century: A Practical Guidebook*. New York: W.W. Norton, 1980. ISBN 0-393-95053-0.

Understanding Music with AI: Perspectives on Music Cognition, edited by Mira Balaban, Kemal Ebcioglu, and Otto Laske. Cambridge, MA: MIT Press; Menlo Park, CA: AAAI Press, 1992. ISBN 0-262-52170-9.

Myanmar

Mranmā–Aṅḡlip abhidhān = Myanmar–English Dictionary. Rankun: Dept. of Myanmar Language Commission, Ministry of Education, Union of Myanmar, 1993.

Compiled and edited by the Myanmar Language Commission.

Mranmā cālui:poṅ:satpui kyam: nhañ. khwaithā:. [Rankun]: 1996.

Translated title: Myanmar orthography treatise.

Roop, D. Haigh. *An Introduction to the Burmese Writing System*. [Honolulu]: Center for Southeast Asian Studies, University of Hawaii at Manoa, 1997. (*Southeast Asia Paper*, 11).

Originally published: New Haven: Yale University Press, 1972. (*Yale linguistic series*). ISBN 0-300-01528-3.

N’Ko

Introduction to N’Ko. <http://home.gwu.edu/~cwme/Nko/Nkohome.htm>

Kanté, Souleymane. *Méthode pratique d’écriture n’ko*, 1961. Kankan, Guinea: Association de tradithérapeutes et pharmacologues, 1995.

N’Ko: The Common Language of Mandens. <http://www.nkoinstitute.com>

N’Ko: The Mandingo Language Site. <http://www.kanjamadi.com>

Ogham

McManus, Damian. *A Guide to Ogam*. Maynooth: An Sagart, 1991. (*Maynooth monographs*, 4). ISBN 1-87068-417-6.

Ol Chiki

Hembram, S. M., et al. *Adibasi Ol script = at’ip’asi al ciki*. Calcutta: Adibasi Socio-Educational & Cultural Association, 1972.

Murmu, Raghunath. *Ranaṛ: A Santali Grammar in Santali*. Singhbhum, Bihar: Adibasi Socio-Educational & Cultural Association, 1972.

Zide, Norman. “Scripts for Munda languages.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York; Oxford: Oxford University Press, 1996. ISBN 0-19-507993-0.

Old Italic

Bonfante, Larissa. “The Scripts of Italy.” In *The World’s Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Cristofani, Mauro. “L’alfabeto etrusco.” In *Lingue e dialetti dell’Italia antica*, a cura di Aldo Larosdocimi. Roma: Biblioteca di storia patria, a cura dell’ Ente per la diffusione e l’educazione storia, 1978. (*Popoli e civiltà dell’Italia antica*, VI.)

Gordon, Arthur E. *Illustrated Introduction to Latin Epigraphy*. Berkeley: University of California Press, 1983. ISBN 0-520-03898-3.

Marinetti, Anna. *Le iscrizioni sudpicene*. I. *Testi*. Firenze: Olschki, 1985. ISBN 88-222-3331-X (v. 1).

Parlangèli, Oronzo. *Studi Messapici*. Milano: Istituto lombardo di scienze e lettere, 1960.

Old Persian

Schmitt, Rüdiger. *The Bisitun Inscriptions of Darius the Great, Old Persian Text*. London, School of Oriental and African Studies, 1991. (*Corpus Inscriptionum Iranicarum*, Part I: *Inscriptions of Ancient Iran*, v.1, Text 1). ISBN 0-7286-0181-8.

Schweiger, Günter. *Kritische Neuedition der achaemenidischen Keilschriften*. Taimering: Schweiger VWT-Verlag, 1998. (*Studien zur Iranistik*). ISBN 3-934548-00-8.

Old South Arabian

Nebes, Norbert, and Peter Stein. “Ancient South Arabian.” In *The Cambridge Encyclopedia of the World’s Ancient Languages*, edited by Roger D. Woodard. 454–487. Cambridge University Press, 2004. ISBN-13: 978-0521562560.

Ryckmans, J. “Origin and Evolution of South Arabian Minuscule Writing on Wood (1).” *Arabian Archaeology and Epigraphy* 12 (2001): 223–235. ISSN 0905-7196.

Smithsonian Institution. “Written in Stone: Inscriptions from the National Museum of Saudi Arabic.”

http://www.mnh.si.edu/epigraphy/figs-stones/x-large/color_xl.jpeg/fig02.jpg

Stein, Peter. “The Ancient South Arabian Minuscule Inscriptions on Wood: A New Genre of Pre-Islamic Epigraphy.” *Jaarbericht van het Vooraziatisch-Egyptisch Genootschap “Ex Oriente Lux”*, 39 (2005): 181–199. ISSN 0075-2118.

Old Turkic

Erdal, Marcel. *A Grammar of Old Turkic*. Leiden & Boston: Brill, 2004. ISBN 9004102949.

Scharlipp, Wolfgang Ekkehard. *Eski Türk run yazıtlarına giriş: ders kitabı = An Introduction to the Old Turkish Runic Inscriptions: A Textbook in English and Turkish*. Engelschoff: Auf dem Ruffel, 2000. ISBN 3-933847-00-X.

von Gabain, A. *Alttürkische Grammatik mit Bibliographie, Lesestücken und Wörterverzeichnis, auch Neutürkisch*. Leipzig: Harrassowitz, 1941. (*Porta Linguarum Orientalium*, 23).

Osmanya

Afkeenna iyo fartiisa: buug koowaad. Xamar: Goosanka afka iyo suugaanta Soomaalida, 1971.

Translated title: *Our language and its handwriting: book one*.

Cerulli, Enrico. “Tentativo indigeno di formare un alfabeto somalo.” *Oriente moderno*, 12 (1932): 212–213. ISSN 0030-5472.

Gaur, Albertine. *A History of Writing*. London: British Library, 1992. ISBN 0-7123-0270-0.

Also published: Rev. ed. New York: Cross River Press, 1992. ISBN 1-558-59358-6.

Gregersen, Edgar A. *Language in Africa: An Introductory Survey*. New York: Gordon and Breach, 1977. (*Library of Anthropology*). ISBN 0-677-04380-5; 0-677-04385-6 (pbk.).

Maino, Mario. “L’alfabeto ‘Osmania’ in Somalia.” *Rassegna di studi etiopici*, 10 (1951): 108–121. ISSN 0390-3699.

Nakanishi, Akira. *Writing Systems of the World: Alphabets, Syllabaries, Pictograms*. Rutland, VT: Tuttle, 1980. ISBN 0-8048-1293-4; 0-8048-1654-9 (pbk.).

Revised translation of *Sekai no moji*.

Phags-pa

Luo, Changpei. *Basibazi yu Yuandai Hanyu [ziliao huibian]* / Luo Changpei, Cai Meibiao bian zhu. Beijing: Kexue chubanshe, 1959.

Poppe, Nikolai Nikolaevich. *The Mongolian Monuments in hP'ags-pa Script*. Translated and edited by John R. Krueger. 2nd ed. Wiesbaden: Harrassowitz, 1957. (*Göttinger asiatische Forschungen*, 8).

Zhaonasiu. *Menggu ziyun jiaoben* / Zhaonasiu, Yang Naisi bian zhu. [Beijing]: Min zu chu ban she, 1987.

Author Zhaonasiu also known as Jagunasutu or Junast.

Philippine Scripts

Doctrina Christiana: The First Book Printed in the Philippines, Manila 1593. A facsimile of the copy in the Lessing J. Rosenwald Collection, with an introductory essay by Edwin Wolf II. Washington, DC: Library of Congress, 1947.

Kuipers, Joel C., and Ray McDermott. "Insular Southeast Asian Scripts." In *The World's Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Santos, Hector. *The Living Scripts*. Los Angeles: Sushi Dog Graphics, 1995. (*Ancient Philippine scripts series*, 2).

User's guide accompanying *Computer Fonts, Living Scripts* software.

Santos, Hector. *Our Living Scripts*. January 31, 1997.

<http://www.bibingka.com/dahon/living/living.htm>

Part of his *A Philippine Leaf*.

Santos, Hector. *The Tagalog Script*. Los Angeles: Sushi Dog Graphics, 1994. (*Ancient Philippine scripts series*, 1).

User's guide accompanying *Tagalog Script Fonts* software.

Santos, Hector. *The Tagalog Script*. October 26, 1996.

<http://www.bibingka.com/dahon/tagalog/tagalog.htm>

Part of his *A Philippine Leaf*.

Phoenician

Branden, Albertus van den. *Grammaire phénicienne*. Beyrouth: Librairie du Liban, 1969. (*Bibliothèque de l'Université Saint-Esprit*, 2).

McCarter, P. Kyle. *The Antiquity of the Greek Alphabet and the Early Phoenician Scripts*. Missoula, MT: Published by Scholars Press for Harvard Semitic Museum, 1975. (*Harvard Semitic Monographs*, 9). ISBN 0-89130-066-X.

Noldeke, Theodor. *Beiträge zur semitischen Sprachwissenschaft*. Strassburg: Karl J. Trübner, 1904.

Reprinted as: vol. 1 of *Beiträge und Neue Beiträge zur semitischen Sprachwissenschaft: achtzehn Aufsätze und Studien*. Amsterdam: APA-Philo Press, [1982].

Also published on microfiche by the American Theological Library Association.

Powell, Barry B. *Homer and the Origin of the Greek Alphabet*. Cambridge, New York: Cambridge University Press, 1991. ISBN 0-521-37157-0.

Reprinted, 1996. ISBN 0-521-58907-X (pbk).

Rejang

Jaspan, M. A. *Folk Literature of South Sumatra: Redjang Ka-Ga-Nga Texts*. Canberra: Australian National University, 1964.

Runic

Friesen, Otto von. *Runorna*. Stockholm: A. Bonnier, [1933]. (*Nordisk kultur*, 6).

Haugen, Einar Ingvald. *The Scandinavian Languages: An Introduction to Their History*. London: Faber, 1976. ISBN 0-571-10423-1.

Also published: Cambridge, MA: Harvard University Press, 1976. ISBN 0-674-79002-2.

Musset, Lucien. *Introduction à la runologie*. Paris: Aubier-Montaigne, 1965.

Page, Raymond Ian. *Runes*. Berkeley: University of California Press; [London]: British Museum, 1987. (*Reading the Past*). ISBN 0-520-06114-4.

British Museum Publications edition has ISBN 0-7141-8065-3.

Samaritan

Ben-Hayyam, Ze'ev. *A Grammar of Samaritan Hebrew, Based on the Recitation of the Law in Comparison with the Tiberian and other Jewish Traditions*. Jerusalem: Hebrew University Magnes Press, 2000. ISBN 1-57506-047-7.

Macuch, Rudolf. *Grammatik des samaritanischen Hebräisch*. Berlin: Walter de Gruyter, 1969. ISBN 9783110083767.

Murtonen, A. *Materials for a Non-Masoretic Hebrew Grammar III: A Grammar of the Samaritan Dialect of Hebrew*. Helsinki: Societas Orientalis Fennica, 1964. (*Studia Orientalia*, 29).

Saurashtra

Učida, Norihiko. *Language of the Saurashtrians in Tirupati*. 2nd revised ed. Bangalore: Mahalaxmi Enterprises, 1991. (In Latin script.)

Učida, Norihiko. *Saurashtra-English Dictionary*. Wiesbaden: Harrassowitz, 1990. ISBN 3447030550. (In Latin script.)

Sharada

Deambi, Kaul and Bushan Kumar. *Śāradā and Ṭākarī Alphabets: Origin and Development*. New Delhi: Indira Gandhi National Centre for the Arts, 2008.

Grierson, George A. "On the Sharada Alphabet." *The Journal of the Asiatic Society of Great Britain and Ireland*, (1916): 677–708.

Shavian

ConScript Unicode Registry [by] John Cowan and Michael Everson. "E700–E72F Shavian."

Included in the ConScript Registry (<http://www.evertype.com/standards/csur/index.html>) in 1997. Shavian was withdrawn from the ConScript Registry in 2001, because of its addition to the Unicode Standard and ISO/IEC 10646.

Crystal, David. *The Cambridge Encyclopedia of Language*. Cambridge, New York: Cambridge University Press, 1987. ISBN 0-521-26438-3.

2nd ed. Cambridge, New York: Cambridge University Press, 1997. ISBN 0-521-55050-5; 0-521-55967-7.

DeMeyere, Ross. *About Shavian*. 1997. <http://www.demeyere.com/Shavian/info.html>.

Shaw, George Bernard. *Androcles and the Lion: An Old Fable Renovated, by Bernard Shaw, with a Parallel Text in Shaw's Alphabet to Be Read in Conjunction Showing Its Economies in Writing and Reading*. Harmondsworth: Penguin Books, 1962.

Sinhala

Gunasekara, Abraham Mendis. *A Comprehensive Grammar of the Sinhalese Language*. New Delhi: Asian Education Services, 1986.

Reprint of 1891 edition.

Sora Sompeng

Mahapatra, Khageshwar. “Soran Sompen’: A Sora Script.” Unpublished conference paper. Delhi, Mysore, 1978–1979.

Zide, Norman. “Scripts for Munda languages.” In *The World's Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Zide, Norman. “Three Munda scripts.” In *Linguistics of the Tibeto-Burman Area*. Vol. 22.2—Fall 1999

Sundanese

Baidillah, Idin, Cucu Komara, and Deus Fitni. *Ngalagena: Panglengkep Pangajaran Aksara Sunda pikeun Murid Sakola Dasar/Dikdas 9 Taun*. [Bandung]: CV Walatra, [2002].

Hardjasaputra, A. Sobana, Tedi Permadi, Undang A. Darsa, and Edi S. Ekadjati. *Rancangan Pembakuan Aksara Sunda*. Bandung, 1998.

Syriac

Kefarnissy, Paul. *Grammaire de la langue Araméenne syriaque*. Beyrouth, 1962.

Nöldeke, Theodor. *Compendious Syriac Grammar*. With a table of characters by Julius Euting. Translated from the 2nd and improved German ed., by James A. Crichton. London: Williams & Norgate, 1904.

Reprinted: Tel Aviv: Zion Pub. Co., [1970].

Robinson, Theodore Henry. *Paradigms and Exercises in Syriac Grammar*. 4th ed. Rev. by L. H. Brockington. Oxford: Clarendon Press; New York: Oxford University Press, 1962. ISBN 0-19-815416-X, 0-19-815458-5 (pbk.).

Tai Le

Coulmas, Florian. *The Blackwell Encyclopedia of Writing Systems*. Oxford, Cambridge: Blackwell, 1996. ISBN 0-631-19446-0.

Dehong writing, pp. 118–119.

Lá ai² mau³ lá ai² ka va³ mi² tse² lau ya pa me na⁴ ka na: tá va l^á na kó ma⁶ sá na² teh ma⁶. Yina⁵ l^{ána} min⁵ su⁴ su⁴ pána² se³ (Yunnan minzu chubanshe). 1988. ISBN 7-5367-1100-4.

Tsa va⁴ má³ hó va³: la ta⁶ mé² sá ai³ seh va² xo ɲa³. Yina⁵lána⁵ mina⁵su⁴ su⁴pána²se³ (Yunnan minzu chubanshe). 1997. ISBN 7-5367-1455-6.

Tai Tham

Peltier, Anatole-Roger. 1996. *Lanna Reader*. Chiang Mai: Wat Tha Kradas.

Kasēm Siriratphiriya, and Mahāwitthayālai Sukhōthaitammāthirāt. *Tūa Mueang: kānrīan phāsā Lānnā phān khrōngsāng kham*. Nonthaburī: Rōngphim Mahāwitthayālai Sukhōthaitammāthirāt, 2548 [2005]. ISBN 974-9942-00-0.

Runggrueangsri, Udom. 2004. *Pacanānukrom Lānnā-Thai: Chabaph maefāhluang*. ISBN 974-685-175-9.

Baephryar phāsā Lānnā. ISBN 974-386-044-4.

Takri

Deambi, Kaul and Bushan Kumar. *Śāradā and Tākārī Alphabets: Origin and Development*. New Delhi: Indira Gandhi National Centre for the Arts, 2008.

Thaana

Geiger, Wilhelm. *Maldivian Linguistic Studies*. New Delhi: Asian Educational Services, 1996. ISBN 81-206-1201-9.

Originally published: Colombo: H. C. Cottle, Govt. Printer, 1919.

Maniku, Hassan Ahmed. *Say It in Maldivian (Dhivehi)*, [by] H. A. Maniku [and] J. B. Disanayaka. Colombo: Lake House Investments, 1990.

Ugaritic

O'Connor, M. "Epigraphic Semitic Scripts." In *The World's Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Walker, C. B. F. *Cuneiform*. London: British Museum Press, 1987. (*Reading the Past*, v. 3.) ISBN 0-7141-8059-9.

University of California Press edition has ISBN 0-520-06115-2 (pbk.).

Vai

Dalby, David. "A Survey of the Indigenous Scripts of Liberia and Sierra Leone: Vai, Mende, Loma, Kpelle and Bassa." *African Language Studies* 8 (1967), 1–51.

Kandakai, Zuke, et al. *Vai kpolo saikilamaa mε = The Standard Vai Script*. Monrovia: University of Liberia African Studies Program, 1962.

Massaquoi, Momolu. "The Vai People and Their Syllabic Writing." *Journal of the Royal African Society* 10.40, July (1911): 459–466.

Singler, John. "Scripts of West Africa." In *The World's Writing Systems*, edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

Stewart, Gail, and P. E. H. Hair. "A Bibliography of the Vai Language and Script." *Journal of West African Languages*, 6.2 (1969): 124.

Yi

Nuo-su bbur-ma shep jie zzit. = *Yi wen jian zi ben*. Chengdu: Sichuan minzu chubanshe, 1984.

Nip huo bbur-ma ssix jie. = *Yi Han zidian*. Chengdu: Sichuan minzu chubanshe, 1990. ISBN 7-5409-0128-4.

R.4 Selected Resources: Technical

American Mathematical Society. *TeX Resources*. <http://www.ams.org/tex/tex-resources.html>

For AMS TeX-related products, see *AMS TeX Resources*. <http://www.ams.org/tex/>

André, Jacques. *Unicode, écriture du monde? / Jacques André, Henri Hudrisier*. Paris: Lavoisier, 2002. (*Document numérique*, v. 6, no. 3–4.) ISBN 2-7426-0594-7.

Bringhurst, Robert. *The Elements of Typographic Style*. 3rd ed. Point Roberts, WA: Hartley & Marks, 2004. ISBN 0-88179-205-5; 0-88179-206-3 (pbk.).

Chaundy, Theodore William. *The Printing of Mathematics: Aids for Authors and Editors and Rules for Compositors and Readers at the University Press, Oxford*. By T. W. Chaundy, P. R. Barrett, and Charles Batey. London: Oxford University Press, [1965].

Reprint of the second impression (revised) 1957.

Deitsch, Andrew. *Java Internationalization* [by] Andrew Deitsch and David Czarnecki. Beijing, Sebastopol, CA: O'Reilly, 2001. ISBN 0-596-00019-7.

Desgraupes, Bernard. *Passeport pour Unicode*. Paris: Vuibert, informatique, 2005. ISBN 2-7117-4827-8.

Developing International Software [by] Dr. International. 2nd ed. Redmond, WA: Microsoft, 2002. ISBN 0-7356-1583-7.

Also published: London: Chrysalis, 2002.

Esselink, Bert. *A Practical Guide to Localization*. Amsterdam, Philadelphia: John Benjamins, 2000. ISBN 1-588-11006-0 (pbk.), 1-588-11005-2 (hardcover).

Rev. ed. of *A Practical Guide to Software Localization*.

Flanagan, David. *Java in a Nutshell*. 5th ed. Sebastopol, CA: O'Reilly, 2005. ISBN 0896007736.

Garneau, Denis. *Keys to Sort and Search for Culturally-Expected Results*. [s.l.] IBM, 1990. (IBM document number GG24-3516, June 1, 1990).

Gillam, Richard. *Unicode Demystified: A Practical Programmer's Guide to the Encoding Standard*. Boston: Addison-Wesley, 2002. ISBN 0-201-70052-2.

Graham, Tony. *Unicode: A Primer*. Foster City, CA: MIS Press, M&T Books, 2000. ISBN 0-7645-4625-2.

The Guide to Translation and Localization: Preparing Products for the Global Marketplace. [5th ed.] Portland, OR: Lingo Systems; [Sandpoint, ID]: Multilingual Computing, 2004. ISBN 0970394829 (pbk.).

IBM. *e-Business Globalization Solution Design Guide: Getting Started*. 2002, updated 2004. (IBM Redbook). (SG24-6851-00). ISBN 0738426563.

Available in PDF, HTML, or hardcopy from:

<http://www.redbooks.ibm.com/abstracts/sg246851.html?Open>

ICU User Guide. <http://icu.sourceforge.net/userguide/>

User guide for International Components for Unicode (ICU), a set of C/C++ and Java libraries for Unicode support.

International Organization for Standardization. *Information Technology—An Operational Model for Characters and Glyphs*. Geneva: 1998. (ISO/IEC TR 15285:1998).

International User Interfaces, edited by Elisa M. del Galdo and Jakob Nielsen. New York: Wiley, 1996. ISBN 0-471-14965-9.

Knuth, Donald E. *T_EX, the Program*. Reading, MA: Addison-Wesley, 1986. (*Computers & Typesetting*, B). ISBN 0-201-13437-3.

Knuth, Donald E. *The T_EXbook*. 21st printing, rev. Reading, MA: Addison-Wesley, 1994. (*Computers & Typesetting*, A). ISBN 0-201-13448-9.

Korpela, Jukka K. *Unicode Explained*. Beijing, Sebastopol, CA: O'Reilly, 2006. ISBN 0-596-10121-X.

Krantz, Steven G. *Handbook of Typography for the Mathematical Sciences*. Boca Raton: Chapman & Hall/CRC, 2001. ISBN 1584881496.

Lamport, Leslie. *L^AT_EX, a Document Preparation System: User's Guide & Reference Manual*. 2nd ed. Reading, MA: Addison-Wesley, 1999, ©1994. ISBN 0-201-52983-1.

Updated for L^AT_EX 2nd ed.

Language Culture Type: International Type Design in the Age of Unicode, edited by John D. Berry; with a special section showing the winners in Bukva:Raz!, the type design competition of the Association typographique internationale. New York: ATypI, Graphis, 2002. ISBN 1-932026-01-0.

Lunde, Ken. *CJKV Information Processing*. 2nd ed. Beijing, Cambridge, MA: O'Reilly, 2009. ISBN 9780596514471 (pbk.).

Mathematics in Type. Richmond, VA: The William Byrd Press, [1954].

PAN Localization Project. *Survey of Language Computing in Asia 2005*, by Sarmad Hussain, Nadir Durrani, and Sana Gul. Lahore: Center for Research in Urdu Language Processing; Ottawa: International Development Research Center, 2005. ISBN 969-8961-00-3.

Available as a book and also in PDF from:

<http://www.idrc.ca/uploads/user-S/11446781751Survey.pdf>

Savourel, Yves. *XML Internationalization and Localization*. Indianapolis, IN: Sams, 2001. ISBN 0-672-32096-7.

Also published: Hemel Hempstead: Prentice-Hall, 2001.

Swanson, Ellen. *Mathematics into Type*. Updated ed. [by] Arlene O'Sean and Antoinette Schleyer. Providence, RI: American Mathematical Society, 1999. ISBN 0821819615.

Takahashi, Tokutaro. "A Proposal For a Standardized Common Use Character Set in East Asian Countries." *Journal of East Asian Libraries*, 63.1 (1980): 48-52. ISSN 1087-5093.

Unicode Guide, [by] Joe Becker, Rich Gillam, Mark Davis, and the Unicode Consortium Editorial Committee. Boca Raton, FL: BarCharts, 2006. (QuickStudy: Computer). ISBN 978-14230-180-9.

Wick, Karel. *Rules for Type-setting Mathematics*. Prague: Publishing House of the Czechoslovak Academy of Sciences, 1965.

W3C Internationalization (I18n) Activity. <http://www.w3.org/International/>

W3C Web Internationalization Tutorials. <http://www.w3.org/International/tutorials/>

Tutorials include *Character sets & encodings in XHTML, HTML and CSS*, and *Using language information in XHTML, HTML and CSS*, both by Richard Ishida.

Yunikodo kanji joho jiten = *Sanseido's Unicode Kanji Information Dictionary* / Yunikodo Kanji Joho Jiten Henshu Iinkai hen. Tokyo: Sanseido, 2000. ISBN 4-385-13690-4.

R.5 Selected Resources: Other

Bibliographies

A Bibliography on Writing and Written Language, edited by Konrad Ehlich, Florian Coulmas, and Gabriele Graefen. Berlin, New York: Mouton de Gruyter, 1996. (*Trends in Linguistics. Studies and Monographs*, 89). ISBN 3-11-010158-0.

Contains references to about 27,500 publications covering mainly 1930–1992.

Michael Everson's Cool Bibliography of Typography and Scripts.

<http://www.evertype.com/scriptbib.html>

Translations

Unicode Consortium. *The Unicode Standard, Version 5.0*. Chinese Simplified language edition. Beijing: Pearson Education Asia Ltd. and Tsinghua University Press, 2010.

In Chinese, authorized partial translation of English language edition of *The Unicode Standard, Version 5.0*. Boston, MA: Addison-Wesley, 2007. ISBN 0-321-48091-0.

Selected Works on Scripts, Languages, and Writing

Allworth, Edward. *Nationalities of the Soviet East: Publications and Writing Systems*. New York: Columbia University Press, 1971. ISBN 0-231-03274-9.

The Alphabet Makers: A Presentation from the Museum of the Alphabet, Waxhaw, North Carolina. 2nd ed. Huntington Beach, CA: Summer Institute of Linguistics, 1991. ISBN 0-938978-13-6.

Alphabete und Schriftzeichen des Morgen- und Abendlandes. 2. überarb. u. erw. Aufl. Berlin: Bundesdruckerei, 1969.

Bergsträsser, Gotthelf. *Introduction to the Semitic Languages: Text Specimens and Grammatical Sketches*. Translated with an appendix on the scripts by Peter T. Daniels. [2nd ed.] Winona Lake: Eisenbrauns, 1995. ISBN 0-931464-10-2.

Translation of *Einführung in die semitischen Sprachen*, 1928.

The Book of a Thousand Tongues, by Eugene A. Nida. Rev. ed. London: United Bible Society, 1972.

First ed. by Eric M. North, 1938.

The Cambridge Encyclopedia of the World's Ancient Languages, edited by Roger D. Woodard. Cambridge, New York: Cambridge University Press, 2004. ISBN 0521562562.

Campbell, George L. *Compendium of the World's Languages*. London: Routledge, 1990. ISBN 0-415-06937-6 (set); 0-415-06978-5 (v.1); 0-415-06979-3 (v.2).

Cleator, Philip Ellaby. *Lost Languages*, by P. E. Cleator. London: Day, 1959.

Also published: London: R. Hale, [1959]; New York: John Day Co., [1961]; [New York]: New American Library, [1962].

Comrie, Bernard, ed. *The Languages of the Soviet Union*. Cambridge: Cambridge University Press, 1981. ISBN 0-521-23230-9; 0-521-9877-6 (pbk.).

Comrie, Bernard, ed. *The World's Major Languages*. Oxford: Oxford University Press, 1987. ISBN 0-19-520521-9; 0-19-506511-5 (pbk.).

Coulmas, Florian. *The Blackwell Encyclopedia of Writing Systems*. Cambridge, MA: Blackwell, 1996. ISBN 0-631-19446-0.

Coulmas, Florian. *The Writing Systems of the World*. Oxford, New York: Blackwell, 1989.

Crystal, David. *The Cambridge Encyclopedia of Language*. 2nd ed. Cambridge, New York: Cambridge University Press, 1997. ISBN 0-521-55050-5; 0-521-55967-7 (pbk.).

Dalby, Andrew. *Dictionary of Languages: The Definitive Reference to More Than 400 Languages*. New York: Columbia University Press, 1998. ISBN 0231115687; 0231115695 (pbk.).

Daniels, Peter T. *The World's Writing Systems*. (See *World's Writing Systems*.)

DeFrancis, John. *Visible Speech: The Diverse Oneness of Writing Systems*. Honolulu: University of Hawaii Press, 1989. ISBN 0-8248-1207-7.

Diringer, David. *The Alphabet: A Key to the History of Mankind*. 3rd ed., completely rev. with the assistance of Reinhold Regensburger. New York: Funk and Wagnalls, 1968.

Also published: London: Hutchinson. ISBN 0-906764-0-8.

Diringer, David. *Writing*. London: Thames and Hudson, 1962.

Also published: New York: Praeger.

Dixon, Robert M. W. *The Languages of Australia*. Cambridge: Cambridge University Press, 1980. ISBN 0-521-22329-6.

Drucker, Johanna. *The Alphabetic Labyrinth: The Letters in History and Imagination*. London: Thames & Hudson, 1999. ISBN 0-500-28068-1.

Endo, Shotoku. *Hayawakari chugoku kantai-ji*. Tokyo: Kokusho Kankokai, Showa 61, [1986].

Faulmann, Carl. *Das Buch der Schrift: enthaltend die Schriftzeichen und Alphabete aller Zeiten und aller Völker*. 2. verm. und verb. Aufl. Wien: Kaiserlich-Königliche Hof- und Staatsdruckerei, 1880.

Reprinted as: *Schriftzeichen und Alphabete aller Zeiten und Völker*. Augsburg: Augustus Verlag, 1990. ISBN 3-8043-0142-8.

Reprinted as: *Das Buch der Schrift: enthaltend die Schriftzeichen und Alphabete aller Zeiten und aller Völker des Erdkreises*. Frankfurt am Main: Eichborn, 1990. ISBN 3-8218-1720-8.

Friedrich, Johannes. *Extinct Languages*. New York: Philosophical Library, 1957.

Translation of *Entzifferung Verschollener Schriften und Sprachen*. Berlin: Springer-Verlag, 1954.

Also published: London: Peter Owen, [1962]; Westport, CT: Greenwood Press, [1971, ©1957]. ISBN 0-8371-5748-X; New York: Dorset Press, 1989, ©1957. ISBN 0-88029-338-1.

Friedrich, Johannes. *Geschichte der Schrift*. Heidelberg: C. Winter, 1966.

Gaur, Albertine. *A History of Writing*. Rev. ed. New York: Cross River Press, 1992. ISBN 1-558-59358-6.

Also published: London: British Library. ISBN 0-7123-0270-0.

Gelb, Ignace J. *A Study of Writing*. Rev. ed. Chicago: University of Chicago Press, 1963. ISBN 0-226-28605-3; 0-226-28606-1 (pbk.).

- Giliarevskii, Rudzhero Sergeevich. *Languages Identification Guide*, by Rudzhero S. Gilyarevsky and Vladimir S. Grivnin. Moscow: Nauka, 1970.
- Gordon, Cyrus Herzl. *Forgotten Scripts: How They Were Deciphered and Their Impact on Contemporary Culture* [by] Cyrus H. Gordon. New York: Basic Books, [1968].
- The Gospel in Many Tongues: Specimens of 875 Languages ...* London: British and Foreign Bible Society, 1965.
- Haarmann, Harald. *Universalgeschichte der Schrift*. Frankfurt: Campus Verlag, 1990. ISBN 3-593-34346-0.
- Habein, Yaeko Sato. *The History of the Japanese Written Language*. Tokyo: University of Tokyo Press, 1984. ISBN 0-86008-347-0; 4-13-087047-5.
- Healey, John F. *The Early Alphabet*. Berkeley: University of California Press; [London]: British Museum, 1990. (*Reading the Past*, 9). ISBN 0-520-07309-6.
British Museum Publications edition has ISBN 0-7141-8073-4.
- A History of Writing: From Hieroglyph to Multimedia*, edited by Anne-Marie Christin. Paris: Flammarion; London: Thames & Hudson, 2002. ISBN 2-08-010887-5.
- Ifrah, Georges. *From One to Zero, a Universal History of Numbers*. New York: Penguin, 1987. ISBN 0-14-009919-0.
Translation of *Histoire universelle des chiffres*. Also published: New York: Viking, 1985. ISBN 0-670-37395-8.
- Isaev, Magomet Izmailovich. *Sto tridtsat' ravnopravnykh: o iazykakh narodov SSSR*. Moskva: Nauka, 1970.
- Jensen, Hans. *Sign, Symbol and Script: An Account of Man's Efforts to Write*. New York: Putnam, 1969.
Translation of *Die Schrift in Vergangenheit und Gegenwart* (Berlin: Deutscher Verlag, 1969). Also published: London: Allen & Unwin. ISBN 0-04-400021-9.
- Katzner, Kenneth. *The Languages of the World*. New ed. London: Routledge, 1995. ISBN 0-415-11809-3.
- Lyovin, Anatole. *Introduction to the Languages of the World*. New York: Oxford University Press, 1997. ISBN 0-19-508115-3; ISBN 0-19-508116-1 (pbk.).
- Malherbe, Michel. *Les Langages de l'humanité: une encyclopédie des 3000 langues parlées dans le monde*. Paris: Laffont, 1995. ISBN 2-221-05947-6.
- Muller, Siegfried H. *The World's Living Languages: Basic Facts of Their Structure, Kinship, Location, and Number of Speakers*. New York: Ungar, 1964.
- Musaev, Kenesbai Musaevich. *Alfavitnyy iazykov narodov SSSR*. Moskva: Nauka, 1965.
- Naik, Bapurao S. *Typography of Devanagari*. 1st ed., rev. Bombay: Directorate of Languages, Govt. of Maharashtra, 1971.
- Nakanishi, Akira. *Writing Systems of the World: Alphabets, Syllabaries, Pictograms*. Rutland, VT: Tuttle, 1980. ISBN 0-8048-1293-4; 0-8048-1654-9 (pbk.).
Revised translation of *Sekai no moji*.
- Nida, Eugene A. *The Book of a Thousand Tongues*. (See *Book of a Thousand Tongues*.)
- Pavlenko, Nikolai Andreevich. *Istoria pis'ma*. 2. izd. Minsk: Vysshaya Shkola, 1987.

Ramsey, S. Robert. *The Languages of China*. 2nd printing with revisions. Princeton: Princeton University Press, 1989. ISBN 0-691-01468-X.

2nd ed. published London: Gollancz, 1974. ISBN 0-575-01758-9.

Robinson, Andrew. *The Story of Writing*. London: Thames and Hudson, 1995. ISBN 0-500-01665-8.

Ruhlen, Merritt. *A Guide to the World's Languages, volume 1: Classification, with a Postscript on Recent Developments*. Stanford, CA: Stanford University Press, 1991. ISBN 0-8047-1894-6 (v. 1).

Also published: London: Arnold. ISBN 0-340-56186-6 (v.1).

Sampson, Geoffrey. *Writing Systems: A Linguistic Introduction*. Stanford, CA: Stanford University Press, 1985. ISBN 0-8047-1254-9.

Also published: London: Hutchinson. ISBN 0-09-156980-X; 0-09-173051-1 (pbk.).

Sekai moji jiten / Kono, Rokuro; Chino, Eiichi; Nishida, Tatsuo. Tokyo: Sanseido, 2001. (*Gengogaku daijiten* = *The Sanseido Encyclopaedia of Linguistics*, 7). ISBN 4-385-15177-6.

Senner, Wayne M. *The Origins of Writing*. Lincoln: University of Nebraska Press, 1989. ISBN 0-8032-4202-6; 0-8032-9167-1 (pbk.).

Shepherd, Walter. *Shepherd's Glossary of Graphic Signs and Symbols*. Compiled and classified for ready reference by Walter Shepherd. New York: Dover, 1971. ISBN 0-486-20700-5.

Also published: London: Dent. ISBN 0-460-03818-4.

Shinmura, Izuru. *Kojien* / Shinmura Izuru hen. Dai 4-han. Tokyo: Iwanami Shoten, 1991.

Passage cited in *Section 12.1*, *Han* is from the 1983 edition, translated from the Japanese by Lee Collins.

Stevens, John. *Sacred Calligraphy of the East*. 3rd ed., rev. and expanded. Boston: Shambala, 1995. ISBN 1-570-62122-5.

Suarez, Jorge A. *The Mesoamerican Indian Languages*. Cambridge: Cambridge University Press, 1983. ISBN 0-521-22834-4; 0-521-29669-2 (pbk.).

von Ostermann, Georg F. *Manual of Foreign Languages*. 4th ed., revised and enlarged. New York: Central Book Company, 1952.

Wemyss, Stanley. *The Languages of the World, Ancient and Modern: The Alphabets, Ideographs, and Other Written Characters of the World in Sound and Symbol*. Philadelphia: 1950.

The World's Writing Systems. Edited by Peter T. Daniels and William Bright. New York: Oxford University Press, 1996. ISBN 0-19-507993-0.

I General Index

The General Index covers the contents of this core specification. To find topics in the Unicode Standard Annexes, Unicode Technical Standards, and Unicode Technical Reports, use the search feature on the Unicode Web site.

For definitions of terms used, see the glossary on the Unicode Web site. To find the code points for specific characters or the code ranges for particular scripts, use the Character Index on the Unicode Web site. (See *Section B.6, Other Unicode Online Resources.*)

A

abbreviation, Coptic 228
abjads 188, 245
abstract character sequences
 definition 67
abstract characters 21
 definition 66
abugidas 189, 190, 277, 363
accent marks *see* diacritics
accented characters
 encoding 9
 Latin 212
 normalization 152
accounting numbers, ideographic 134
acrophonic numerals 151, 226
Aegean numbers 473
Afrikaans 216
Ainu 429
Aiton 373
Alchemical Symbols 527
 reference materials 622
Algonquian 457
Ali Gali 441
aliases
 character name 66, 136, 574
 property 123
 property value 123
allocation areas 34
allocation of encoded characters 33–40, 601
Alphabetic (informative property) 140
alphabets 188
 European 211–243
 mathematical 498–502
Alpine 468
alternate format characters (deprecated) . . 141, 554–555
Amharic 438
Ancient Symbols 530
angle brackets (U+2329 and U+232A)
 deprecated for technical publication 518
Annexes, Unicode Standard (UAX) xxvii, 586
 as components of Unicode Standard 59
 conformance 64
 list of 64
annotation characters 563–564
 use in plain text discouraged 564

ANSI/ISO C
 wchar_t and Unicode 148
apostrophe (U+0027) 200
Arabic 250–266
 digits 503
Arabic-Indic digits 253–254
 signs used with 255
ArabicShaping.txt 256, 260, 271
Aramaic 277, 324, 355, 441, 478
archaic scripts 465–474
areas of the Unicode Standard 34
ARIB 523
Armenian 232–233
arrows 515–516
ASCII
 characters with multiple semantics 193
 transparency of UTF-8 27
 Unicode modeled on 1
 zero extension 148, 597
Assamese 296
assigned code points 8, 23
Athapascan 457
atomic character boundaries 159
Avestan 482
 reference materials 622

B

Balinese 394–399
 reference materials 623
Bamum 455–456
 reference materials 623
Bangla 295–299
base characters 238
 definition 79
 multiple 45
 ordered before combining marks 161, 238
Basic Multilingual Plane (BMP) 1, 33
 allocation areas 37
 representation in UTF-16 27
Basque 216
Batak 402–403
 reference materials 623
benefits of Unicode 1
Bengali 295–299
Bidi Class (normative property) 130
Bidi Mirrored (normative property) 135
Bidi Mirroring Glyph (informative property) 135

- BidiMirroring.txt 135
 - Bidirectional Algorithm, Unicode 40, 63
 - bidirectional ordering 15
 - controls 141, 553
 - bidirectional text 40, 63
 - Middle Eastern scripts 245
 - nonspacing marks in 163
 - punctuation in 192
 - big-endian 30
 - definition 63
 - Bihari 293
 - binary comparison and sort order
 - caution for UTF-16 27
 - UTF differences 168, 170
 - UTF-8 29
 - blocks of the Unicode Standard 34, 187
 - Blocks.txt 34
 - BMP *see* Basic Multilingual Plane
 - BNF (Backus-Naur Form) 583
 - BOCU-1 *see* UTXN #6, BOCU-1
 - MIME-Compatible Unicode Compression
 - Bodhi 325
 - Bodo 292
 - BOM (U+FEFF) 30, 50, 98–100, 561–562
 - Bopomofo 426–427
 - boundaries, text 8, 46, 140, 158–159, 167
 - see also* UAX #14, Unicode Line Breaking Algorithm
 - see also* UAX #29, Unicode Text Segmentation
 - boustrophedon 41, 470
 - Brahmi 277, 324, 355, 359–361, 364
 - reference materials 623
 - Braille 534–536
 - Breton 216
 - Buginese 393–394
 - Buhid 392
 - Bulgarian 230
 - bullets 202
 - numeric 505
 - Burmese *see* Myanmar
 - Byelorussian 230
 - byte order mark (BOM) (U+FEFF) .. 30, 50, 98–100, 561–562
 - byte ordering
 - changing 61
 - conformance 62
 - byte serialization 30, 50
 - Byzantine Musical Symbols 540
- C**
- C language
 - wchar_t and Unicode 148
 - C0 and C1 control codes 23, 139, 544
 - Cambodian *see* Khmer
 - camelcase 174
 - Canadian Aboriginal Syllabics 457–458
 - reference materials 624
 - candrabindu 294, 349
 - canonical composite characters
 - see* canonical decomposable characters
 - canonical composition algorithm 104
 - canonical decomposable characters
 - definition 88
 - canonical decomposition 48
 - definition 88
 - mappings 87
 - canonical equivalence
 - definition 88
 - nonspacing marks 164
 - canonical equivalent character sequences
 - conformance 60, 61
 - canonical mappings
 - see* canonical decomposition mappings
 - canonical ordering algorithm 104
 - canonical precomposed characters
 - see* canonical decomposable characters
 - Cantonese 413
 - capital letters 124, 172, 211
 - Carian 474
 - reference materials 624
 - carriage return (U+000D) (CR) 154, 545
 - carriage return and line feed (CRLF) 154
 - case 217
 - and text processes 9
 - beyond ASCII 172
 - camelcase 174
 - case folding 175
 - case operations (conformance) 64, 115–120
 - case operations and normalization 177
 - case operations, reversibility 174
 - cased (definition) 115
 - case-insensitive comparison ... 119, 168, 169, 175
 - casing context (definition) 116
 - conversion 117
 - detection 118
 - European alphabets 211
 - exceptional Latin pairs 214, 217
 - Georgian 234
 - lowercase 124, 172, 211
 - mapping tables 146
 - mappings 115, 126, 172–174
 - mappings noted in code charts 575
 - titlecase 124, 172
 - Turkish I 173, 174, 214
 - uppercase 124, 172, 211
 - see also* default case
 - Case (normative property) 124, 172
 - CaseFolding.txt 126, 175, 176
 - caseless letters 217
 - Catalan 215
 - cedilla 213
 - CEF *see* character encoding forms
 - CES *see* character encoding schemes
 - CESU-8
 - see* UTR #26, Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)
 - Chakma 350–351
 - reference materials 624
 - Cham 390–391
 - reference materials 624
 - character encoding forms (CEF) 24–29, 597
 - see also* Unicode encoding forms
 - character encoding model 24, 31
 - see also* UTR #17, Unicode Character Encoding Model
 - character encoding schemes (CES) 30–32
 - see also* Unicode encoding schemes

- character encoding standards
 - coverage by Unicode 2
- Character Index 590
- character literals, Unicode
 - code point notation U+ 583
- character mapping
 - interchange format *see* UTS #22, Character Mapping Markup Language (CharMapML)
- character names 65, 135–139, 599
 - aliases 66, 136, 574
 - conventions 581
 - for CJK ideographs 577
 - for control codes 138, 139
 - in code charts 571–574
 - matching 136
- character properties
 - see* properties
 - see also individual properties, e.g.* Combining Class
- character semantics 1, 60, 64–65, 599
 - as Unicode design principle 14
 - ASCII 193
 - definition 65
- character sequences
 - abstract *see* abstract character sequences
 - canonical equivalent *see* canonical equivalent character sequences
 - compatibility equivalent *see* compatibility equivalent character sequences
 - conformance 60
 - named 136
- character sequences, combining 79
- character shaping selectors (deprecated) 555
- character tabulation (U+0009) 545
- characters
 - abstract *see* abstract characters
 - arrangement in Unicode 35
 - assigned 8, 23
 - blocks 34, 187
 - boundaries 158
 - canonical decomposable *see* canonical decomposable characters
 - classes 583
 - code charts 571–579, 589
 - coded *see* encoded characters
 - combining *see* combining characters
 - compatibility decomposable *see* compatibility decomposable characters
 - composite *see* decomposable characters
 - concept of 11, 46
 - conformance definitions 66–69
 - confusable 179
 - conversion 145–147
 - decomposable *see* decomposable characters
 - deprecated *see* deprecated characters
 - encoded *see* encoded characters
 - encoding forms *see* encoding forms
 - encoding schemes *see* encoding schemes
 - end-user perceived 46
 - format control 23, 51, 193, 543–569
 - glyphs, relationship to 11
 - graphic 23
 - identity (definition) 65
 - ignored in processing 180–185
 - interpretation 59
 - layout control 51, 545–554
 - modification 61
 - names list 571–574
 - names *see* character names
 - not encoded in Unicode 2
 - number encoded in this and earlier versions 601
 - number encoded in Version 6.1 2
 - precomposed *see* decomposable characters
 - properties *see* properties
 - semantics *see* character semantics
 - special 50, 543–569
 - supplementary *see* supplementary characters
 - transcoding 145–147
 - unsupported 148–149
- characters, not glyphs
 - in spoofing 180
 - Unicode principle 11
- CharMapML
 - see* UTS #22, Character Mapping Markup Language (CharMapML)
- charsets
 - IANA registered names 31
- charts, character code *see* code charts
- Cherokee 456
 - reference materials 624
- Chinese 413–414
 - Cantonese 413
 - Hakka 427
 - Mandarin 413
 - Minnan (Hokkien/Fujian, incl. Taiwanese) 427
 - simplified and traditional 413
- Chu hán 412
- Chu Nôm 607
- citations for
 - properties 58
 - Unicode algorithms 58
 - Unicode Standard 57
- CJK ideographs 190, 406–421
 - accounting numbers 134
 - CJK Compatibility Ideographs 420–421
 - CJK Compatibility Supplement 421
 - CJK Strokes 423, 609
 - CJK Unified Ideographs 406–420
 - CJK Unified Ideographs Extension A 410
 - CJK Unified Ideographs Extension B 420
 - CJK Unified Ideographs Extension C 420
 - CJK Unified Ideographs Extension D 420
 - code charts 577
 - compatibility ideographs in Plane 2 40
 - component structure 416
 - encoding blocks 409
 - ideographic description sequences 423–425
 - ideographic variation mark (U+303E) 425
 - KangXi radicals 419, 421–422
 - names 577
 - numbers 503
 - numeric values 133, 151
 - order of encoding 418
 - radicals 421–422
 - source standards 407–409
 - unknown or unavailable 208
 - Vietnamese 405
- CJK Miscellaneous Area 38
- CJK punctuation and symbols 207
 - compatibility forms 209
 - overscores and underscores 209

- quotation marks198
- sesame dots208
- vertical forms209
- CJK-JRG (Chinese/Japanese/Korean Joint Research Group)606
- CJKV Ideographs Area38
- CLDR (Unicode Common Locale Data Repository)590
- cluster boundaries158
- code charts 571–579, 589
 - representative glyphs572
- code point sequences
 - notation582
- code points5, 22
 - assigned8, 23
 - assignment35, 601
 - categories22
 - default ignorable149, 184
 - definition67
 - designated23
 - notation581
 - number in Unicode Standard1
 - private-use *see* private-use code points
 - reserved *see* reserved code points
 - semantics24
 - surrogate *see* surrogates
 - unassigned *see* unassigned code points
 - undesignated23
- code positions *see* code points
- code set independence14
- code unit sequences
 - definition89
 - ill-formed (definition)91
 - notation582
 - well-formed (definition)91
- code units
 - definition89
 - isolated89
- code values *see* code units
- coded character representations
 - see* coded character sequences
- coded character sequences
 - definition67
- coded characters *see* encoded characters
- codespace *see* Unicode codespace
- coeng375, 376
- Collation Algorithm, Unicode (UCA)10
- collation *see* sorting
- collation tables146
- combining character sequences42, 79
 - defective163
 - definition80
 - Latin212
 - line breaking160
 - matching160
 - order of base character and marks161, 238
 - rendering160
 - selection158
 - truncation 161–162
- combining characters 41–46, 82–86, 159–166
 - blocking reordering551
 - canonical ordering 47, 104, 126
 - class zero127
 - combining marks 238–239
 - definition79
 - dependence238
 - display order43
 - keyboard input160
 - ligatures45
 - multiple43
 - multiple base characters45
 - normalization of152
 - ordering conventions42
 - rendering of marks162–166
 - reordrant127
 - script-specific42
 - split128
 - strikethrough130
 - subjoined129
 - typographical interaction43, 126
 - vertical stacking44
 - see also* diacritics
- Combining Class (normative property)126
- combining classes102, 126, 165
 - class zero characters126
 - definition102
- combining grapheme joiner (U+034F)551
- combining half marks141, 243
- combining marks *see* combining characters
- comma below213
- Compatibility and Specials Area20, 38
- compatibility characters18
- compatibility composite characters21
 - see* compatibility decomposable characters
- compatibility decomposable characters20
 - definition87
- compatibility decomposition48
 - definition87
- compatibility decomposition mappings87
- Compatibility Encoding Scheme for UTF-16
 - see* UTR #26, Compatibility Encoding Scheme for UTF-16: 8-Bit (CESU-8)
- compatibility equivalence
 - definition88
- compatibility equivalent character sequences
 - conformance61
- compatibility mappings
 - see* compatibility decomposition mappings
- compatibility precomposed characters
 - see* compatibility decomposable characters
- compatibility variants20
 - mapping177
- composite characters
 - see* decomposable characters
 - compatibility *see* compatibility decomposable characters
- Composition Exclusion (normative property)74
- compression153
 - see also* UTS #6, A Standard Compression Scheme for Unicode (SCSU)
- conferences590
- conformance55–120
 - clause and definition updates603
 - definitions64–69
 - examples51
 - ISO/IEC 10646 implementations600
 - requirements59–63
- confusables179
- conjunct consonants
 - Indic158, 281

Myanmar	369
selection of clusters	158
contextual shaping	
apostrophe	200
Arabic	251
not used for Hebrew final forms	247
quotation marks	197
Syriac	269
contour tones	237
control codes	23, 51, 544
graphics for	517
names	139
properties	545
semantics	24, 545
specified in Unicode	545
control sequences	544
conversion of characters	96, 145–147, 185
convertibility	
as Unicode design principle	19
Coptic	225, 227–229
reference materials	624
corporate use subarea	558
corrigenda	57
CR (U+000D carriage return)	154, 545
CRLF (carriage return and line feed)	154
Croatian	216
digraphs	216
culturally expected sorting	10, 168
Cuneiform	
Old Persian	484
Sumero-Akkadian	485–487
Ugaritic	483
Cuneiform and Hieroglyphic Area	39
currency symbols	494–496
encoded in script blocks	495
cursive joining	548–551
Arabic	256–262
control characters for	141, 252, 443, 547
Mandaic	480
Mongolian	442–444
N’Ko	453
Syriac	269–272
transparency	550
cursive scripts	245
Cypriot	474
reference materials	629
<i>see also</i> Linear B	
Cyrillic	229–231
Czech	216
D	
danda, in Devanagari block	291
Danish	215
dashes	195
Database, Unicode Character	
<i>see</i> Unicode Character Database (UCD)	
dead consonants, Indic	281
dead keys	160
decomposable characters	48
definition	86
normalization of	152
decomposition	48, 86–88
canonical <i>see</i> canonical decomposition	
compatibility <i>see</i> compatibility decomposition	
definition	86
in normalization	152
mapping, definition	86
mappings noted in code charts	576
default case	
algorithms	64, 115–120
conversion	117
detection	118
folding	117
default caseless matching	119
default grapheme clusters	159
<i>see also</i> UAX #29, Unicode Text Segmentation	
Default Ignorable Code Point (property)	184
default ignorable code points	149, 184
default property values	72
definition	72
defective combining character sequences	163
definition	81
dependent vowel signs	
Indic	280
Khmer	378
Philippine scripts	392
deprecated characters	55, 574
alternate format	141, 554–555
definition	68
Derived Age (property)	149
derived properties	
definition	77
DerivedCoreProperties.txt	115, 124, 184
DerivedNormalizationProps.txt	177
Deseret	459–460
reference materials	625
design goals of Unicode	3
design principles of Unicode	10–19
designated code points	23
Devanagari	278–295
Dhivehi	274
diacritics	42, 238
alternative glyphs	212, 239
Czech	212
display in isolation	45, 195, 239
double	85, 141, 240
Greek	222–223, 226
Latin	212–214
Latvian	213
mathematical	501
on i and j	214
rendering	162–166
Slovak	212
spacing clones of	237, 239
symbol	42, 242
<i>see also</i> combining characters	
dictionary symbols	524
digit form names	254
digits	151
Arabic	503
Arabic-Indic	253–254
compatibility	504
decimal	133
glyph variants	505
hexadecimal	504
Myanmar	503
national shapes	555
Shan	503
superscript and subscript	504

Tai Tham503
 digraphs 216, 219, 220
 dingbats 526–527
 directionality15, 40
 East Asian scripts406
 Middle Eastern scripts245
 Mongolian442
 musical symbols537
 normative property130
 Ogham467
 Old Italic468
 Philippine scripts393
 Runic470
 discussion list for Unicode590
 Dogri292
 Domino Tiles527
 dotless i 173, 174, 214
 dotted circle
 in code charts80, 239
 in fallback rendering163
 to indicate diacritic41
 to indicate vowel sign placement43
 double diacritics 85, 141, 240
 Dutch215, 216
 dynamic composition
 as Unicode design principle18
 Dzongkha325

E

East Asian scripts 405–435
 writing direction41
see also CJK ideographs
 Eastern Arabic-Indic digits253
 EBCDIC
 newline function154
see UTR #16, UTF-EBCDIC
 editing, text boundaries for 158–159
 efficiency
 as Unicode design principle11
 Egyptian hieroglyphs 487–491
 reference materials625
 e-mail discussion list for Unicode590
 emoji522, 523
 animal symbols525
 cultural symbols525
 zodiacal symbols525
 Emoticons525
 Enclosed Alphanumerics533
 enclosing marks243
 definition80
 encoded characters5, 22
 allocation 33–40, 601
 definition67
 encoding form conversion
 definition95
 encoding forms24–29
 ISO/IEC 10646 definitions597
 encoding forms, Unicode
see Unicode encoding forms
 encoding model for Unicode characters24, 31
see also UTR #17, Unicode Character Encoding Model
 encoding schemes30–32

encoding schemes, Unicode
see Unicode encoding schemes
 endian ordering
see byte order mark (BOM) (U+FEFF)
 end-user subarea559
 English215
 equivalent sequences152
 as Unicode design principle18
 case-insensitivity169, 175
 combining characters in matching160
 conformance61
 Hangul syllables432
 in sorting and searching167
 language-specific88
 security implications179
see also canonical equivalence
see also compatibility equivalence
see also encoding forms, encoding schemes
 errata xxix, 57, 590
 escape sequences544
 not used in Unicode1, 3
 Esperanto216
 Estonian216
 Ethiopic438–440
 reference materials625
 Etruscan467
 euro sign (U+20AC)496
 European alphabetic scripts211–243
 eyelash-RA286

F

fallback rendering184
 of nonspacing marks162
 FAQ (Frequently Asked Questions)590
 Faroese215
 Farsi250, 252
 featural syllabaries189
 FF (U+000C form feed)154, 545
 file separator (U+001C)545
 Finnish215
 Finno-Ugric Transcription (FUT)
see Uralic Phonetic Alphabet (UPA)
 fixed-width Unicode encoding form (UTF-32) .26, 93
 flat tables146
 Flemish215
 fonts
 and Unicode characters13
 for mathematical alphabets500–502
 style variation for symbols493
 form feed (U+000C) (FF)154, 545
 format control characters23, 51, 193, 543–569
 deprecated554–555
 prefixed141
 stateful553
 fraction characters511
 fraction slash (U+2044)200, 509
 French216
 Frisian216
 FTP site, Unicode Consortium589
 fullwidth forms in East Asian encodings429–430
 futhark470

G

Garshuni266

- Ge'ez 438
 - General Category (normative property) 130
 - list of values 130
 - general punctuation 191–209
 - General Scripts Area 38
 - geometrical symbols 520–522
 - Georgian 233–235
 - German 215
 - geta mark (U+3013) 208
 - Glagolitic 231–232
 - reference materials 625
 - Glossary 590
 - glyph selection tables 146
 - glyphs 5, 12
 - characters, relationship to 11
 - diacritics alternative 212, 239
 - Greek alternative 223–224
 - Latin alternative 212
 - mathematical alternative 512
 - missing 184
 - representative in code charts 572
 - standardized variants 556
 - symbols alternative 493
 - golden numbers 471
 - Gothic 471–472
 - reference materials 626
 - grapheme base 238
 - definition 81
 - grapheme clusters 8, 46
 - see also* UAX #29, Unicode Text Segmentation
 - default 159
 - definition 81
 - grapheme extender
 - definition 81
 - grapheme joiner, combining (U+034F) 551
 - graphic characters 23
 - Greek 222–226
 - acrophonic numerals 151, 226
 - alternative glyphs 223–224
 - ancient musical notation 540–542
 - editorial marks 205, 626
 - letters as symbols 223–225, 512
 - see also* Cypriot, Linear B
 - Greenlandic 216
 - group separator (U+001D) 545
 - guillemets 198
 - Gujarati 303–304
 - Gurmukhi 300–303
- H**
- Hakka 427
 - halant 277
 - see also* virama
 - half marks, combining 141, 243
 - half-consonants, Indic 282
 - halfwidth forms in East Asian encodings ... 429–430
 - Han ideographs *see* CJK ideographs
 - Han unification 414–420
 - and language tags 157
 - history 605–607
 - language usage 412
 - source separation rule 410, 415
 - source standards 407–409
 - Hangul Area 38
 - Hangul syllables 405, 430–433
 - and combining marks 86
 - as grapheme clusters 46
 - canonical decomposition 109
 - collation 433
 - composition 110
 - conjoining jamo 107–114
 - equivalent sequences 432
 - Hangul Compatibility Jamo 431
 - Hangul Jamo 430–433
 - Hangul Syllables block 432–433
 - Johab set 432
 - name generation 111
 - normalization 431
 - standard 108
 - Hangzhou numerals 508
 - Hanja *see* CJK ideographs
 - Hanunóo 392
 - Hanzi *see* CJK ideographs
 - harakat, Arabic pronunciation marks 250
 - hasant 296
 - hash tables 146
 - Hebrew 246–250
 - hentaigana 429
 - hieroglyphs
 - Egyptian 487–491
 - Meroitic 491–492
 - high surrogate
 - definition 88
 - high-surrogate code points 59, 559
 - high-surrogate code units 88
 - higher-level protocols
 - definition 68
 - Hindi 278
 - Hiragana 428
 - historic scripts 465–474
 - horizontal tab (U+0009) 545
 - HTML newline function 155
 - Hungarian 216
 - hyphenation 547
 - as a text process 8
 - hyphens 195, 547
- I**
- I Ching symbols 529
 - IANA charset names 31
 - Icelandic 215
 - identifiers 167
 - see also* UAX #31, Unicode Identifier and Pattern Syntax
 - Ideographic (informative property) 140
 - ideographic description sequences 424
 - Ideographic Rapporteur Group (IRG) 407, 606
 - Ideographic Variation Database *see* UTS #37, Unicode Ideographic Variation Database
 - ideographs *see also* CJK ideographs
 - IDNA *see* UTS #46, Unicode IDNA Compatibility Processing
 - IICore 411, 607
 - ill-formed
 - definition 91
 - Imperial Aramaic 478–479
 - reference materials 626
 - implementation guidelines 145–185

- in a Unicode encoding form
 - definition 92
 - in-band mechanisms 568
 - Indian rupee sign (U+20B9) 496
 - Indic scripts 277–322, 323–325
 - principles, in terms of Devanagari 279–285
 - relation to ISCII standard 278
 - Indonesian 215
 - industry character sets
 - covered in Unicode 2
 - information separators (U+001C..U+001F) 545
 - informative properties
 - definition 74
 - Inscriptional Pahlavi 481
 - Inscriptional Parthian 481
 - inside-out rule 162
 - interchange restrictions 23
 - International Phonetic Alphabet (IPA) 188, 218–219
 - reference materials 627
 - Spacing Modifier Letters 236
 - see also* phonetic alphabets
 - internationalization 14
 - Internationalization & Unicode Conference 590
 - Internet protocols
 - UTF-8 as preferred encoding 28
 - Inuktitut 457
 - invisible operators 516
 - iota subscript 223
 - IPA *see* International Phonetic Alphabet
 - IRG (Ideographic Rapporteur Group) 407, 606
 - Irish 215, 466
 - ISCII standard and Unicode 278
 - ISO/IEC 10646 593–600
 - conformance of Unicode implementations 599
 - encoding forms 597
 - synchrony with Unicode Standard 598
 - timeline compared to Unicode versions 594
 - Italian 215
 - ITC Zapf Dingbats 526
 - IUC *see* Internationalization & Unicode Conference
- J**
- jamos *see* Hangul syllables
 - Japanese 405
 - Javanese 399–401
 - reference materials 627
 - Jawi 264
 - jihvamuliya 295, 349
 - Johab 432
 - joiners 252
 - combining grapheme joiner (U+034F) 551
 - word joiner (U+2060) 546
 - zero width joiner (U+200D) 252, 549
 - justification 164
- K**
- Kaithi 345–347
 - reference materials 627
 - Kana (Hiragana and Katakana) 428–429
 - Kanbun 421
 - KangXi radicals 419, 421–422
 - Kanji *see* CJK ideographs
 - Kannada 315–317
 - Kashmiri 293
 - Katakana 428–429
 - Kawi 394, 396
 - Kayah Li 389–390
 - reference materials 627
 - KC (normalization form)
 - see* Normalization Form KC
 - KD (normalization form)
 - see* Normalization Form KD
 - keytop labels 517
 - Khamti Shan 372
 - Kharoshthi 355–356
 - reference materials 628
 - Khmer 374–383
 - characters not recommended 380
 - syllable components, order of 381
 - killer 189
 - Batak 402
 - Brahmi 359
 - Meetei Mayek 352
 - Myanmar (asat) 370
 - see also* virama
 - Konkani 292
 - Korean Hangul *see* Hangul
 - Kurdish 250, 264
- L**
- Ladino 246
 - language tags 157, 565–568
 - and Han unification 157
 - use strongly discouraged 568
 - Lanna 384
 - Lao 366–368
 - last-resort glyphs 184
 - Latin 212–222
 - alternative glyphs 212
 - Basic Latin 215
 - encoding blocks 34
 - IPA Extensions 218–219
 - Latin Extended Additional 220–222
 - Latin Extended-A 216
 - Latin Extended-B 216–218
 - Latin Extended-C 220
 - Latin Extended-D 221
 - Latin Ligatures 220
 - Latin-1 Supplement 215
 - Phonetic Extensions 219–221
 - Latvian 216, 221
 - cedilla 213
 - layout control characters 51, 545–554
 - leading surrogates
 - see* high-surrogate code units
 - legibility criterion for plain text 15
 - Lepcha 335–336
 - reference materials 628
 - letter spacing 547
 - letterlike symbols 496–502
 - LF (U+000A line feed) 154, 545
 - ligatures 548–551
 - Arabic 258–259
 - combining characters on 45
 - control characters for 141
 - for nonspacing marks 165
 - Latin 220
 - selection 159

Syriac	272
Limbu	342–344
reference materials	628
line breaking	153–156, 546–548
control characters	143
in South Asian scripts	366, 371, 383
recommendations	155
<i>see also</i> UAX #14, Unicode Line Breaking Algorithm	
line feed (U+000A) (LF)	154, 545
line separator (U+2028) (LS)	154, 547
line tabulation (U+000B) (VT)	545
Linear B	473
reference materials	629
<i>see also</i> Cypriot	
linear boundaries	159
Lisu	461–463
reference materials	629
Lithuanian	216
little-endian	30
definition	63
Locale Data Markup Language	
<i>see</i> UTS #35, Unicode Locale Data Markup Language (LDML)	
logical order	
as Unicode design principle	15
exceptions to	128
logograph	190
logosyllabaries	190
low surrogate	
definition	88
low-surrogate code points	59, 559
low-surrogate code units	88
lowercase	124, 172, 211
LS (U+2028 line separator)	154, 547
Lycian	474
reference materials	630
Lydian	474
reference materials	630

M

MacOS newline function	154
Mahjong Tiles	527
mail discussion list for Unicode	590
Maithili	292
major version	56
Malay	215
Malayalam	317–322
Maltese	216
Manchu	441
Mandaic	479–481
reference materials	630
Mandarin	413
Manden	450
map symbols	524
mapping tables <i>see</i> tables of character data	
Marathi	278, 286, 290
markup languages	
and Unicode conformance	568
line breaking	153
<i>see also</i> UTR #20, Unicode in XML and Other Markup Languages	
Mathematical (informative property)	511

mathematical expression format characters	141
<i>see also</i> UTR #25, Unicode Support for Mathematics	
mathematical symbols	511–516
alphabets	498–502
alphanumeric	497–502
fonts	500–502
format characters	516
fragments for typesetting	518
invisible operators	516
operators	512–513
reference materials	630
standardized variants	516
MathML	513
matras	127, 280
Meetei Mayek	351–353
reference materials	630
Meroitic	
cursive	491–492
hieroglyphs	491–492
reference materials	630
Miao	463–464
reference materials	631
Middle Eastern scripts	245–275
Min	413
Minnan (Hokkien/Fujian, incl. Taiwanese)	427
minor version	56
minus sign	513
commercial (U+2052)	203
mirrored property	
<i>see</i> Bidi Mirrored (normative property)	
mirroring of paired punctuation	197
Miscellaneous Symbols	523
missing glyphs	184
modifier letters	235–238
Modifier Letters, Spacing	220
Mongolian	337, 440–447
writing direction	442
multibyte encodings	
compared to UTF-8	28
multistage tables	146
musical symbols	536–542
ancient Greek	540–542
Balinese	398
Byzantine	540
directionality	537
Gregorian	537
reference materials	631
Western	536–539
Myanmar	368–373
digits	503
Myanmar Extended-A	371
reference materials	632

N

N’Ko	450–454
reference materials	632
named character sequences	136
names, character <i>see</i> character names	
namespace	66
NEL (U+0085 next line)	154, 545
Nepali	278
neutral directional characters	130
New Tai Lue	384–385

- newline function (NLF)154, 545
 newline guidelines 153–156
 next line (U+0085) (NEL)154, 545
 NFC (Normalization Form C)47
 NFD (Normalization Form D)47
 NFKC (Normalization Form KC)47
 NFKD (Normalization Form KD)47
 NLF (newline function)154, 545
 no-break space (U+00A0)546
 base for diacritic in isolation 45, 195, 239
 no-break space, narrow (U+202F)445
 noncharacter code points *see* noncharacters
 noncharacters23, 50, 560
 conformance59
 definition68
 handling61
 in code charts574
 interchange restrictions24
 semantics24
 U+10FFFF (not a character code)560
 U+FDD0..U+FDEF23, 560
 U+FFFE (not a character code)50, 560
 U+FFFF (not a character code)23, 560
 nondecomposable characters48
 non-joiner, zero width (U+200C)252, 549
 nonlinear boundaries159
 non-overlap principle in Unicode encoding forms .24
 nonspacing marks238
 definition80
 display in isolation 45, 195, 239
 positioning165
 rendering 162–166
 see also combining characters
 see also diacritics
 normalization47, 152
 and case operations177
 canonical ordering algorithm 47, 104, 126
 conformance63
 of private-use characters558
 see also UAX #15, Unicode Normalization Forms
 stability101
 Normalization Form C (NFC)47
 Normalization Form D (NFD)47
 Normalization Form KC (NFKC)47
 Normalization Form KD (NFKD)47
 normalization forms 101–107
 definition106
 specification103
 normative behaviors
 definition65
 normative properties
 definition73
 list74
 may change74
 Norwegian215
 notational conventions 581–584
 notational systems191
 nukta265, 287
 null (U+0000)
 as Unicode string terminator545
 number forms
 CJK ideographs151
 numbers
 handling151
 ideographic accounting134
 numerals502–509
 acrophonic226
 Chinese counting rods510
 Coptic229
 Cuneiform487
 Ethiopic439
 Greek acrophonic151
 Hangzhou508
 old-style201
 Roman151, 511
 Rumi507
 Suzhou-style508
 numeric separators203
 numeric shape selectors (deprecated)555
 Numeric Type (normative property)133
 Numeric Value (normative property)133
 numero sign (U+2116)496
- O**
- object replacement character (U+FFFC)564
 octet583
 Ogham466–467
 reference materials632
 Ol Chiki353–354
 reference materials632
 Old Italic467–469
 reference materials632
 Old Persian484–485
 reference materials633
 Old South Arabian475–477
 reference materials633
 Old Turkic472
 reference materials633
 old-style numerals201
 Oriya304–306
 Oromo438
 Osmanya447
 reference materials633
 out-of-band mechanisms568
 overlapping encodings24
 overscores201
- P**
- Pahlavi, Inscriptional481
 reference materials626
 Panjabi300
 paragraph or section marks203
 paragraph separator (U+2029) (PS)154, 547
 Parthian, Inscriptional481
 reference materials626
 Pashto250
 Persian250, 252
 Phags-pa336–341
 reference materials634
 Phaistos Disc symbols530
 Phake373
 Philippine scripts392–393
 reference materials634
 Phoenician477
 reference materials634
 phonemes190
 phonetic alphabets188
 IPA Extensions218–219
 Phonetic Extensions219–221

Spacing Modifier Letters 236–238
 Uralic Phonetic Alphabet (UPA) 203, 219
see also International Phonetic Alphabet (IPA)

Pinyin 215

pivot code, Unicode as 146

plain text
 as Unicode design principle 14
 legibility criterion 15

planes of Unicode codespace 33
 Plane 0 (BMP) 33
 Plane 1 (SMP) 33, 39
 Plane 14 (SSP) 33
 Plane 2 (SIP) 33, 40
 Planes 15-16 (Private Use) 40, 559

Playing Cards 528

points, Hebrew pronunciation marks 246

policies of the Unicode Consortium 590

Polish 216

Portuguese 215

precomposed characters
see decomposable characters
 compatibility *see* compatibility decomposable
 characters

prefixed format control characters 141

Private Use Area (PUA) 38, 558

Private Use planes 34, 40, 559

private-use characters
 properties 557
 semantics 24

private-use code points 23, 148
 conformance 60
 definition 78
 high surrogates 559

processing code, choice of Unicode encoding form 28

properties 14, 70–78, 121–143
 aliases 123
 aliases (definition) 78
 and Unicode algorithms 74
 data tables 146
 derived *see* derived properties
 in Unicode Character Database (UCD) 34
 informative *see* informative properties
 normative references to 58, 63
 normative *see* normative properties
 of control codes 545
 provisional *see* provisional properties
 simple *see* simple properties
*see also individual properties, e.g. combining
 classes*

property values
 aliases 123
 aliases (definition) 78
 default 72
 default (definition) 72
 normative references to 63

PropertyAliases.txt 78, 583
 PropertyValueAliases.txt 78, 583
 PropList.txt 126

Provençal 216

provisional properties
 definition 75

PS (U+2029 paragraph separator) 154, 547

PUA (Private Use Area) 38, 558

pulli 306

punctuation 191–209
 blocks containing 187
 CJK 207
 doubled 201
 in bidirectional text 192
 paired 197
 small form variants 209
 typographic forms 192
 vertical forms 209

Punctuation and Symbols Area 38

Punjabi 300

Q

quotation marks 197–199
 East Asian 199
 European 198

R

radicals, KangXi and other CJK 421–422

radical-stroke index 419

record separator (U+001E) 545

recycling symbols 524

referencing 63
 properties 58
 Unicode algorithms 58
 Unicode Standard 57

regional indicator symbols 534

regular expressions 156
 and line breaking 153
see also UTS #18, Unicode Regular Expressions

Rejang 401–402
 reference materials 635

rendering of text 5, 8, 13
 fallback 184
 unsupported characters 149

repertoire of abstract characters 22

replacement character (U+FFFD) . 32, 51, 62, 96, 185,
 565

reserved code points 23, 148
 definition 68
 in code charts 574
 preservation in interchange 24
see also unassigned code points

Rhaeto-Romanic 216

rich text 14

right single quotation mark (U+2019)
 preferred for apostrophe 200

right-to-left text 40
 East Asian scripts 406
 Middle Eastern scripts 245

roadmap for script additions 34

Roman numerals 151, 511

Romanian 216
 comma below 213

Romany 216

Rumi numeral forms 507

Runic 469–471
 reference materials 635

rupee sign, Indian (U+20B9) 496

Russian 229

S

- Samaritan 273–274
 - reference materials 635
- Sami 216
- Sanskrit 278
- Saurashtra 347–348
 - reference materials 635
- scalar values, Unicode
 - see* Unicode scalar values
- scripts
 - in Unicode Standard 2
 - roadmap for future additions 34
 - types of 191
 - see also* UAX #24, Unicode Script Property
- SCSU
 - see* UTS #6, A Standard Compression Scheme for Unicode
- searching 167–169
 - as a text process 8
 - case-insensitive 169, 175
- section or paragraph marks 203
- security issues 179
- self-synchronization of encoding forms 25
- semantics
 - see* character semantics
- sequences
 - notation 582
- Serbian
 - corresponding digraphs in Croatian 216
- Shan 383
 - digits 503
- Sharada 348–349
 - reference materials 635
- Shavian 461
 - reference materials 635
- Show Hidden 61, 163, 184, 557
- SHY (U+00AD soft hyphen) 547
- Sibe 442
- signature for Unicode data 51, 561–562
- simple properties
 - definition 77
- simplified Chinese 413
- Sindhi 250, 292
- Sinhala 324–325
 - reference materials 636
- SIP (Supplementary Ideographic Plane) 33, 40
- slash, fraction (U+2044) 200
- Slovak 216
- Slovenian 216
- small letters 124, 172, 211
- SMP (Supplementary Multilingual Plane) 33, 39
- soft hyphen (U+00AD) (SHY) 547
- Somali 447
- Sora Sompeng 354
 - reference materials 636
- Sorbian 216
- sorting 10, 167
 - and combining grapheme joiner 552
 - as a text process 8
 - case-insensitive 168
 - culturally expected 10, 168
 - language-insensitive 168
 - see also* Unicode Collation Algorithm (UCA)
- source separation rule 410, 415
- South Asian scripts 277–322, 323–344
- Southeast Asian scripts 363–393
- space (U+0020)
 - base for diacritic in isolation 46, 195, 239
- space characters 194, 546–548
 - graphics for 517
- space, zero width (U+200B) 194
- spacing clones of diacritics 237, 239
- spacing marks 238
 - definition 80
- Spacing Modifier Letters 236–238
- Spanish 215
- special characters 50, 543–569
- SpecialCasing.txt 115, 126
- Specials 561–565
- spell-checking
 - as a text process 8
- spellings, alternative
 - see* equivalent sequences
- spoofing 179
- SSP (Supplementary Special-purpose Plane) 33
- stability 76, 122
 - as Unicode design principle 18
- stacked boundaries 159
- stacking sequences 43
 - nondefault 44
- Standard Compression Scheme for Unicode (SCSU)
 - see* UTS #6, A Standard Compression Scheme for Unicode
- standardized variants 444, 556
 - mathematical symbols 516
- StandardizedVariants.txt 444, 516
- standards coverage 2
- starters 103
- stateful encoding
 - not used in Unicode 3
 - paired format controls 553
- string comparison 10
- string literals, Unicode
 - code point notation `\u1234` 583
- strings, Unicode 32, 90
 - null termination 545
- strong directional characters 130
- styled text 14
- sublinear searching 169
- subsets, supported 53
 - conformance 60
 - ISO/IEC 10646 specification for 599
- substitution character
 - see* replacement character
- Sumero-Akkadian 485–487
- Sundanese 403–404
 - reference materials 636
- superscripts 236
 - and subscripts 510
- supplementary characters
 - in UTF-16 strings 32
 - tables for 146
- Supplementary General Scripts Area 38
- Supplementary Ideographic Plane (SIP) 33, 40
- Supplementary Multilingual Plane (SMP) 33, 39
- supplementary planes
 - representation in UTF-8 28
 - representation in UTF-16 27
- Supplementary Private Use Areas 40, 559

Supplementary Special-purpose Plane (SSP)	33
supported subsets	53
conformance	60
supralineation	228
surrogate code points	
<i>see</i> surrogates	
surrogate pairs	27, 93
definition	89
processing	28, 149–151
surrogates	23, 88–89, 559
interchange restrictions	23
isolated surrogates, handling	32
isolated surrogates, ill-formed	93
isolated surrogates, uninterpreted	89
support levels	150
Surrogates Area	38, 559
Suzhou-style numerals	508
svasti signs	331
Swahili	215
Swedish	215
syllabaries	188
alphabetic property	140
featural	189
Syoti Nagri	344–345
symbols	493–542
animal	525
appearance variation	493
arrows	515–516
cultural	525
currency	494–496
dictionary	524
dingbats	526–527
emoji	522, 523, 534
Enclosed Alphanumerics	533
fragments for mathematical typesetting	518
game	524
gender	524
genealogical	524
geometrical	520–522
Khmer lunar calendar	383
letterlike	496–502
map	524
mathematical	511–516
mathematical alphanumeric	497–502
miscellaneous	523
musical	536–542
numerals	502–509
recycling	524
regional indicator	534
technical	517–520
weather	524
zodiacal	525
symmetric swapping format characters (deprecated)	555
Syriac	266–272
reference materials	636
T	
tab (U+0009 character tabulation)	545
tab, vertical (U+000B)	154, 545
tables of character data	145–147
optimization	146
supplementary characters	146
tag characters	565–569
Tagalog	392
Tagbanwa	392
tags, language	157, 565–568
use strongly discouraged	568
Tai Le	383–384
reference materials	636
Tai Tham	385–387
digits	503
reference materials	637
Tai Viet	387–389
Tai Xuan Jing symbols	529
Takri	349–350
reference materials	637
Tamil	306–313
TCHAR in Win32 API	148
Technical Notes (UTN)	589
Technical Reports (UTR)	586
abstracts	587
Technical Standards (UTS)	xxviii, 586
abstracts	586
technical symbols	517–520
Telugu	313–315
terminal emulation	494
text boundaries	8, 46, 140, 158–159, 167
<i>see also</i> UAX #14, Unicode Line Breaking Algorithm	
rithm	
<i>see also</i> UAX #29, Unicode Text Boundaries	
text elements	5, 8, 158
boundaries	167
for sorting	168
variable-width nature	29
text processes	4, 8–10
text rendering	5, 8, 13
text selection, boundaries for	158–159
Thaana	274–275
reference materials	637
Thai	364–366
Tibetan	325–334
Tifinagh	448
Tigre	438
tilde (U+007E)	203
titlecase	124, 172
Todo	441
tone letters	237–238
tone marks	
Bopomofo spacing	426, 427
Chinantec	238
Chinese	237
Tai Le	383
Thai	364
Vietnamese	214
traditional Chinese	413
traffic signs	524
trailing surrogates	
<i>see</i> low-surrogate code units	
transcoding	145–147
tables	146
Transport and Map Symbols	525
triangulation in transcoding	146
tries	146
truncation	
combining character sequences	161–162
surrogates and	151

- Turkish216
 - case mapping of I 173, 174, 214
 - cedilla213
 - two-stage tables146
- U**
- U+ notation583
- U+10FFFF (not a character code)560
- U+FEFF (BOM) 561–562
- U+FFFE (not a character code)560
- U+FFFF (not a character code)560
- UAX (Unicode Standard Annex) xxvii, 586
 - as component of Unicode Standard59
 - conformance64
 - list of64
- UCA *see* Unicode Collation Algorithm
- UCD *see* Unicode Character Database
- UCS (Universal Character Set)
 - see* ISO/IEC 10646
- UCS-2597
- UCS-4597
- Ugaritic 483–484
 - reference materials637
- Uighur337, 441
- Ukrainian230
- unassigned code points23, 59, 149
 - defined as reserved code points68
 - handling55
 - properties of72
 - semantics59
 - see also* reserved code points
- underscores201
- undesignated code points23
- Unicode 1.0 Name (informative property)139
- Unicode algorithms
 - and properties74
 - conformance63
 - definition69
 - normative references to58, 63
- Unicode Bidirectional Algorithm16, 40
 - see also* UAX #9, Unicode Bidirectional Algorithm
- Unicode Character Database (UCD) . xxviii, 122, 590
 - as component of Unicode Standard59
 - changes56
 - properties in34
- Unicode character encoding model24, 31
 - see also* UTR #17, Unicode Character Encoding Model
- Unicode character literals
 - code point notation U+583
- Unicode codespace
 - allocation numbers601
 - definition67
 - planes33
 - size1, 22
- Unicode Collation Algorithm (UCA)10
 - see also* UTS #10, Unicode Collation Algorithm
- Unicode Common Locale Data Repository (CLDR)
 -590
- Unicode conferences590
- Unicode Consortium585
 - addresses591
 - Consortium membership in standards bodies 585
 - e-mail discussion list590
- FTP site589
- membership585
- policies590
- Web site589
- Unicode data signature51, 561–562
- Unicode data types147–148
 - for C147–148
- Unicode encoding forms89–95
 - advantages of each28
 - conformance26, 62
 - definition90
 - fixed-width (UTF-32)26, 93
 - signatures562, 563
 - variable-width27, 93, 94
 - see also* encoding forms
- Unicode encoding schemes
 - conformance97–101
 - definition97
 - endian ordering30
 - see also* encoding schemes
- Unicode escape sequence notation \u1234583
- Unicode Regular Expressions *see* UTS #18, Unicode Regular Expressions
- Unicode scalar values
 - definition89
- Unicode security mechanisms
 - see also* UTS #39, Unicode Security Mechanisms
 - Unicode security179
- Unicode Standard
 - allocation of encoded characters33–40
 - architecture7–10
 - areas34
 - benefits1
 - blocks34, 187
 - code charts571–579, 589
 - components59
 - conformance55–120
 - conformance of ISO/IEC 10646 implementations
 -600
 - corrections57
 - definitions for conformance64–69
 - design goals3
 - design principles10–19
 - errata57, 590
 - normative references to57, 63
 - number of characters2, 601
 - number of code points1, 22
 - script coverage2
 - security issues179
 - synchrony with ISO/IEC 10646598
 - updates590
 - versions *see* versions of the Unicode Standard
 - see also* Version 6.1
- Unicode Standard Annexes (UAX) xxvii, 586
 - as components of Unicode Standard59
 - conformance64
 - list of64
- Unicode string literals
 - code point notation \u1234583
- Unicode strings32
 - definition90
- Unicode Technical Committee (UTC)585
- Unicode Technical Notes (UTN)589
- Unicode Technical Reports (UTR)586
 - abstracts587

- Unicode Technical Standards (UTS)xxviii, 586
 - abstracts 586
 - UnicodeData.txt 115, 126
 - unification
 - as Unicode design principle 17
 - see also* Han unification
 - Unified Repertoire and Ordering (URO) 415, 606
 - see also* Han unification
 - Unihan Database 122, 418, 419, 577, 590, 607
 - Unihan.zip 75, 122
 - unit separator (U+001F) 545
 - Universal Character Set (UCS)
 - see* ISO/IEC 10646
 - universality
 - as Unicode design principle 10
 - Unix
 - and UTFs 29
 - newline function 154
 - UTF-8 in 14
 - UTF-32 in 27
 - unsupported characters 148–149
 - upadhmaniya 295, 349
 - update version 57
 - uppercase 124, 172, 211
 - Uralic Phonetic Alphabet (UPA) 203, 219
 - Urdu 250
 - URO (Unified Repertoire and Ordering) 415, 606
 - see also* Han unification
 - UTF, Unicode Transformation Formats 24, 90
 - advantages of each 28
 - as encoding form or scheme 100
 - binary comparison and sort order differences
 - 168, 170
 - in APIs 148
 - UTF-8 27, 94, 598
 - ASCII transparency 27
 - binary comparison and sort order 29
 - bit distribution (table) 94
 - BOM in 98, 101, 561
 - byte ranges 94
 - compared to multibyte encodings 28
 - encoding form (definition) 94
 - encoding scheme 30
 - encoding scheme (definition) 98
 - in Unix 14
 - in UTF-16 order 170
 - non-shortest form is invalid 94, 179
 - preferred encoding for Internet protocols 28
 - security and 179
 - signature 98, 101, 561
 - UTF-16 27, 93, 598
 - binary comparison and sort order caution 27
 - bit distribution (table) 93
 - BOM in 98, 561
 - encoding form (definition) 93
 - encoding scheme (definition) 98
 - encoding schemes 30
 - in ISO/IEC 10646 598
 - in UTF-8 order 171
 - surrogates and string handling 32, 149
 - UTF-16BE (Big-endian) 562
 - encoding scheme 30
 - encoding scheme (definition) 98
 - UTF-16LE (Little-endian) 562
 - encoding scheme 30
 - encoding scheme (definition) 98
 - UTF-32 26, 93
 - BOM in 99
 - encoding form (definition) 93
 - encoding scheme (definition) 99
 - encoding schemes 30
 - in Unix 27
 - UTF-32BE (Big-endian)
 - encoding scheme 30
 - encoding scheme (definition) 99
 - UTF-32LE (Little-endian)
 - encoding scheme 30
 - encoding scheme (definition) 99
 - UTF-EBCDIC
 - see* UTR #16, UTF-EBCDIC
 - UTN (Unicode Technical Note) 589
 - UTR (Unicode Technical Report) 586
 - abstracts 587
 - UTS (Unicode Technical Standard) xxviii, 586
 - abstracts 586
- ## V
- Vai 454–455
 - reference materials 637
 - valid (synonym for well-formed) 92
 - variable-width Unicode encoding form 27, 93, 94
 - variants
 - compatibility 20
 - fullwidth and halfwidth 209
 - mathematical symbols 516
 - small form 209
 - standardized 556
 - variation selectors 142, 556
 - ideographic variation mark (U+303E) 425
 - Mongolian free variation selectors 444
 - variation sequences 556
 - for Phags-pa 340–341
 - Version 6.1 59
 - number of characters 2, 601
 - versions of the Unicode Standard xxviii, 55, 590, 601–602
 - backward compatibility 55
 - compared to ISO/IEC 10646 editions 601
 - content 56
 - interaction in implementations 149
 - numbering 56
 - property changes 56
 - stability 56
 - updates 590
 - vertical tab (U+000B) 154, 545
 - vertical text 41, 192, 209
 - East Asian scripts 406
 - Mongolian 442
 - Vietnamese 214, 220
 - ideographs 405
 - virama 189, 277
 - definition 280
 - Kharoshthi 358
 - Khmer 376
 - Myanmar 369
 - Philippine scripts 392
 - virama-like characters 142

visual order used for Thai and Lao	16	ZWNJ <i>see</i> zero width non-joiner (U+200C)
vowel harmony		ZWSP <i>see</i> zero width space (U+200B)
Mongolian	445	
vowel marks, Middle Eastern scripts	245	
vowel separator		
Mongolian	446	
vowel signs		
Indic	43, 280	
Khmer	378	
Philippine scripts	392	

W

wchar_t	
and Unicode encoding forms	28
in C language	148
weak directional characters	130
weather symbols	524
Web site, Unicode Consortium	589
Weierstrass elliptic function symbol	497
well-formed	
definition	91
Welsh	216
Where Is My Character?	591
wide characters	
data type in C	148
wiggly fence (U+29DB)	514
Windows newline function	154
word breaks	160, 546–548
in South Asian scripts	366, 371, 383
word joiner (U+2060)	546
writing direction <i>see</i> directionality	
writing systems	188–191
Wu (Shanghainese)	413

X

Xibe	442
Xishuang Banna Dai	384
XML	
<i>see</i> UTR #20, Unicode in XML and Other Markup Languages	

Y

yen currency sign	495
Yi	433–435
reference materials	638
Yiddish	246
Yijing Hexagram Symbols	529
ypogegrammeni	223
yuan currency sign	495

Z

Zapf Dingbats	526
zero extension relation among encodings	597
zero width joiner (U+200D)	252, 549
zero width no-break space (U+FEFF)	50, 63, 546
initial	100, 562
zero width non-joiner (U+200C)	252, 549
zero width space (U+200B)	546
for word breaks in South Asian scripts	366, 371, 383
zero-width space characters	547
ZWJ <i>see</i> zero width joiner (U+200D)	
ZWNBSP <i>see</i> zero width no-break space (U+FEFF)	