Technical Reports

L2/24-031

Proposed Update Unicode® Standard Annex #44

UNICODE CHARACTER DATABASE

Version	Unicode 16.0.0
Editors	Ken Whistler (ken@unicode.org)
Date	2023-11-08
This Version	https://www.unicode.org/reports/tr44/tr44-33.html
Previous Version	https://www.unicode.org/reports/tr44/tr44-32.html
Latest Version	https://www.unicode.org/reports/tr44/
Latest Proposed Update	https://www.unicode.org/reports/tr44/proposed.html
Revision	33

Summary

This annex provides the core documentation for the Unicode Character Database (UCD). It describes the layout and organization of the Unicode Character Database and how it specifies the formal definitions of the Unicode Character Properties.

Status

*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time. Publication does not imply endorsement by the Unicode Consortium. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.*

***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.*

Please submit corrigenda and other comments with the online reporting form [Feedback]. Related information that is useful in understanding this annex is found in Unicode Standard Annex #41, "Common References for Unicode Standard Annexes." For the latest version of the Unicode Standard, see [Unicode]. For a list of current Unicode Technical Reports, see [Reports]. For more information about versions of the Unicode Standard, see [Versions]. For any errata which may apply to this annex, see [Errata].

Contents

Note: *the information in this annex is not intended as an exhaustive description of the use and interpretation of Unicode character properties and behavior. It must be used in conjunction with the data in the other files in the Unicode Character Database, and relies on the notation and definitions supplied in The Unicode Standard. All chapter references are to Version 16.0.0 of the standard unless otherwise indicated.*

1 Introduction

The Unicode Standard is far more than a simple encoding of characters. The standard also associates a rich set of semantics with each encoded character—properties that are required for interoperability and correct behavior in implementations, as well as for Unicode conformance. These semantics are cataloged in the Unicode Character Database (UCD), a collection of data files which contain the Unicode character code points and character names. The data files define the Unicode character properties and mappings between Unicode characters (such as case mappings).

This annex describes the UCD and provides a guide to the various documentation files associated with it. Additional information about character properties and their use is contained in the Unicode Standard and its annexes. In particular, implementers should familiarize themselves with the formal definitions and conformance requirements for properties detailed in *Section 3.5, Properties* in [Unicode] and with the material in *Chapter 4, Character Properties* in [Unicode]. Additional discussion about the Unicode character property model can be found in [UTR23].

The latest version of the UCD is always located on the Unicode website at:

https://www.unicode.org/Public/UCD/latest/

The specific files for the UCD associated with this version of the Unicode Standard (16.0.0) are located at:

https://www.unicode.org/Public/16.0.0/

Stable, archived versions of the UCD associated with all earlier versions of the Unicode Standard can be accessed from:

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

For a description of the changes in the UCD for this version and earlier versions, see the UCD Change History.

2 Conformance

The Unicode Character Database is an integral part of the Unicode Standard.

The UCD contains normative property and mapping information required for implementation of various Unicode algorithms such as the Unicode Bidirectional Algorithm, Unicode Normalization, and Unicode Casefolding. The data files also contain additional informative and provisional character property information.

Each specification of a Unicode algorithm, whether specified in the text of [Unicode] or in one of the Unicode Standard Annexes, designates which data file(s) in the UCD are needed to provide normative property information required by that algorithm.

For information on the meaning and application of the terms, *normative*, *informative*, *contributory*, and *provisional*, see *Section 3.5, Properties* in [Unicode].

For information about the applicable terms of use for the UCD, see the Unicode Terms of Use.

2.1 Simple and Derived Properties

2.1.1 Simple Properties

Some character properties in the UCD are simple properties. This status has no bearing on whether or not the properties are normative, but merely indicates that their values are not derived from some combination of other properties.

2.1.2 Derived Properties

Other character properties are derived. This means that their values are derived by rule from some other combination of properties. Generally such rules are stated as set operations, and may or may not include explicit exception lists for individual characters.

Certain simple properties are defined merely to make the statement of the rule defining a derived property more compact or general. Such properties are known as contributory properties. Sometimes these contributory properties are defined to encapsulate the messiness inherent in exception lists. At other times, a contributory property may be defined to help stabilize the definition of an important derived property which is subject to stability guarantees.

Derived character properties are not considered second-class citizens among Unicode character properties. They are defined to make implementation of important algorithms easier to state. Included among the first-class derived properties important for such implementations are: Uppercase, Lowercase, XID_Start, XID_Continue, Math, and Default_Ignorable_Code_Point, all defined in DerivedCoreProperties.txt, as well as derived properties for the optimization of normalization, defined in DerivedNormalizationProps.txt.

Implementations should simply use the derived properties, and should not try to rederive them from lists of simple properties and collections of rules, because of the chances for error and divergence when doing so.

Definitions of property derivations are provided for information only, typically in comment fields in the data files. Such definitions may be refactored, refined, or corrected over time. These definitions are presented in a modified set notation, expressed as set additions and/or subtractions of various other property values. For example:

```
# Derived Property: ID_Start
#  Characters that can start an identifier.
#  Generated from:
#      Lu + Ll + Lt + Lm + Lo + Nl
#    + Other_ID_Start
#    - Pattern_Syntax
#    - Pattern_White_Space
```

When interpreting definitions of derived properties of this sort, keep in mind that set subtraction is not a commutative operation. Thus "Lo + Lm - Pattern_Syntax" defines a different set than "Lo - Pattern_Syntax + Lm". The order of property set operations stated in the definitions affects the composition of the derived set.

If there are any cases of mismatches between the definition of a derived property as listed in DerivedCoreProperties.txt or similar data files in the UCD, and the definition of a derived property as a set definition rule, the explicit listing in the data file should *always* be taken as the normative definition of the property. As described in Stability of Releases the property listing in the data files for any given version of the standard will never change for that version.

2.1.3 Properties Dependent on External Specifications

In limited cases, a Unicode character property defined in the Unicode Character Database may have an external dependency on another specification which is not a part of the Unicode Standard, and whose data is not formally part of the UCD. In such cases, version stability for the UCD is attained by requiring that dependency to be based on a known, published version of the external specification.

Starting with Version 10.0 of the UCD and continuing through Version 12.1, the clear example of such an external dependency was the derivation of some segmentation-related character properties, in part based on emoji properties associated with UTS #51, "Unicode Emoji" [UTS51]. The details of the derivation were described in the respective annexes, [UAX14] and [UAX29], as well as in the documentation portions of the associated UCD property files. See [Data14] and [Props]. The version of UTS #51 used for those segmentation properties in each of the relevant versions of the UCD was clearly identified in those annexes and data files. Starting with Version 13.0 of the UCD, however, the emoji properties which the UCD previously depended on have been formally incorporated into the UCD, so that they no longer constitute an external dependency.

An external dependency may impact either a simple or a derived property.

2.2 Use of Default Values

Unicode character properties have default values. Default values are the value or values that a character property takes for an unassigned code point, or in some instances, for designated subranges of code points, whether assigned or unassigned. For example, the default value of a binary Unicode character property is always "N".

For the formal discussion of default values, see D26 in *Section 3.5, Properties* in [Unicode]. For conventions related to default values in various data files of the UCD and for documentation regarding the particular default values of individual Unicode character properties, see Default Values.

2.3 Stability of Releases

Just as for the Unicode Standard as a whole, each version of the UCD, once published, is absolutely stable and will *never* change. Each released version is archived in a directory on the Unicode website, with a directory number associated with that version. URLs pointing to that version's directory are also stable and will be maintained in perpetuity.

Any errors discovered for a released version of the UCD are noted in [Errata], and if appropriate will be corrected in a *subsequent* version of the UCD.

Stability guarantees constraining how Unicode character properties can (or cannot) change between releases of the UCD are documented in the Unicode Consortium Stability Policies [Stability].

2.3.1 Changes to Properties Between Releases

Updates to character properties in the Unicode Character Database may be required for any of three reasons:

1. To cover new characters added to the standard
2. To add new character properties to the standard
3. To change the assigned values for a property for some characters already in the standard

While the Unicode Consortium endeavors to keep the values of all character properties as stable as possible between versions, occasionally circumstances may arise which require changing them. In particular, as less well-documented scripts, such as those for minority languages, or historic scripts are added to the standard, the exact character properties and behavior may not fully be known when the script is first encoded. The properties for some of these characters may change as further information becomes available or as implementations turn up problems in the initial property assignments. As far as possible, any readjustment of property values based on growing implementation experience is made to be compatible with established practice.

All changes to normative or informative property values, to the status or type of a property, or to property or property value aliases, must be approved by an explicit decision taken by the Unicode Technical Committee. Changes to provisional property values are subject to less stringent oversight.

Occasionally, a character property value is changed to prevent incorrect generalizations about a character's use based on its nominal property values. For example, U+200B ZERO WIDTH SPACE was originally classified as a space character (General_Category=Zs), but it was reclassified as a Format character (General_Category=Cf) to clearly distinguish it from space characters in its function as a format control for line breaking.

There is no guarantee that a particular value for an enumerated property will actually have characters associated with it. Also, because of changes in property value assignments between versions of the standard, a property value that once had characters associated with it may later have none. Such conditions and changes are rare, but implementations must not assume that all property values are associated with non-null sets of characters. For example, currently the special Script property value Katakana_Or_Hiragana has no characters associated with it.

2.3.2 Obsolete Properties

In some instances an entire property may become *obsolete*. For example, the ISO_Comment property was once used to keep track of annotations for characters used in the production of name lists for ISO/IEC 10646 code charts. As of Unicode 5.2.0 that property became obsolete, and its value is now defaulted to the null string for all Unicode code points.

An obsolete property is never removed from the UCD.

2.3.3 Deprecated Properties

Occasionally an obsolete property may also be formally *deprecated*. This is an indication that the property is no longer recommended for use, perhaps because its original intent has been replaced by another property or because its specification was somehow defective. See also the general discussion of Deprecation.

A deprecated property is never removed from the UCD.

Table 1 lists the properties that are formally deprecated as of this version of the Unicode Standard.

Table 1. Deprecated Properties

Property Name	Deprecation Version	Reason
Grapheme_Link	5.0.0	Duplication of ccc=9
Hyphen	6.0.0	Supplanted by Line_Break property values
ISO_Comment	6.0.0	No longer needed for chart generation; otherwise not useful
Expands_On_NFC	6.0.0	Less useful than UTF-specific calculations
Expands_On_NFD	6.0.0	Less useful than UTF-specific calculations
Expands_On_NFKC	6.0.0	Less useful than UTF-specific calculations
Expands_On_NFKD	6.0.0	Less useful than UTF-specific calculations
FC_NFKC_Closure	6.0.0	Supplanted in usage by NFKC_Casefold; otherwise not useful

2.3.4 Stabilized Properties

Another possibility is that an obsolete property may be declared to be *stabilized*. Such a determination does not indicate that the property should or should not be used; instead it is a declaration that the UTC (Unicode Technical Committee) will no longer actively maintain the property or extend it for newly encoded characters. The property values of a stabilized property are frozen as of a particular release of the standard.

A stabilized property is never removed from the UCD.

Table 2 lists the properties that are formally stabilized as of this version of the Unicode Standard.

Table 2. Stabilized Properties

Property Name	Stabilization Version
Hyphen	4.0.0
ISO_Comment	6.0.0

3 Documentation

This annex provides the core documentation for the UCD, but additional information about character properties is available in other parts of the standard and in additional documentation files contained within the UCD.

3.1 Character Properties in the Standard

The formal definitions related to character properties used by the Unicode Standard are documented in *Section 3.5, Properties* in [Unicode]. Understanding those definitions and related terminology is essential to the appropriate use of Unicode character properties.

See *Section 4.1, Unicode Character Database*, in [Unicode] for a general discussion of the UCD and its use in defining properties. The rest of Chapter 4 provides important explanations regarding the meaning and use of various normative character properties.

3.2 The Character Property Model

For a general discussion of the property model which underlies the definitions associated with the UCD, see Unicode Technical Report #23, "The Unicode Character Property Model" [UTR23]. That technical report is informative, but over the years various content from it has been incorporated into normative portions of the Unicode Standard, particularly for the definitions in Chapter 3.

UTR #23 presents the important distinction between properties defined for strings (in contrast to properties defined for characters or code points) and character properties that have values that are strings. The latter are referred to as *string-valued properties* in UTR #23 and in this annex. UTR #23 also discusses string functions and their relation to character properties.

3.3 NamesList.html

NamesList.html formally describes the format of the NamesList.txt data file in BNF. That data file is used to drive the PDF formatting of the Unicode code charts and names list. See also *Section 24.1, Character Names List*, in [Unicode] for a detailed discussion of the conventions used in the Unicode names list as formatted for the online code charts.

3.4 StandardizedVariants.html

StandardizedVariants.html has been obsoleted as of Version 9.0 of the UCD. This file formerly documented standardized variants, showing a representative glyph for each. It was closely tied to the data file, StandardizedVariants.txt, which defines those sequences normatively.

The function of StandardizedVariants.html to show representative glyphs for standardized variants has been superseded. There are now better means of illustrating the glyphs. Many standardized variation sequences are shown in the Unicode code charts directly, in summary sections at the ends of the names list for any block which contains them. Glyphs for standardized variants of CJK compatibility ideographs are also shown directly in the Unicode code charts.

3.5 Emoji Variation Sequences

Emoji variation sequences are a special class of variation sequences involving emoji characters. They are divided into two subtypes: an *emoji presentation sequence*, consisting of an emoji character base followed by the variation selector U+FE0F, and a *text presentation sequence*, consisting of an emoji character base followed by the variation selector U+FE0E. Such sequences come in pairs: the text presentation sequence shown with a black and white presentation, as seen in the Unicode code charts, and the emoji presentation sequence shown with a colorful icon, as usually seen in implementations on mobile devices and elsewhere.

Starting with Version 9.0.0, the following page in the Unicode emoji subsite area shows appropriate representative glyphs for all emoji variation sequences, with separate columns for text presentation sequences and for emoji presentation sequences:

https://www.unicode.org/emoji/charts/emoji-variants.html

The data file which defines the exact list of emoji variation sequences is emoji-variation-sequences.txt. That file is maintained in the UCD, but emoji variation sequences are documented in Unicode Technical Standard #51, *Unicode Emoji* [UTS51].

3.6 Unihan and UAX #38

Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38] describes the format and content of the Unihan Database [Unihan], which collects together all property information for CJK unified ideographs. That annex also specifies in detail which of the Unihan character properties are normative, informative, or provisional.

The Unihan Database contains extensive and detailed mapping information for CJK unified ideographs encoded in the Unicode Standard, but it is aimed *only* at those ideographs, not at other characters used in the East Asian context in general. In contrast, East Asian legacy character sets, including important commercial and national character set standards, contain many non-CJK characters. As a result, the Unihan Database must be supplemented from other sources to establish mapping tables for those character sets.

The majority of the content of the Unihan Database is released for each version of the Unicode Standard as a collection of Unihan data files in the UCD. Because of their large size, these data files are released only as a zipped file, Unihan.zip. The details of the particular data files in Unihan.zip and the CJK properties each one contains are provided in [UAX38]. For versions of the UCD prior to Version 5.2.0, all of the CJK properties were listed together in a very large, single file, Unihan.txt.

3.7 UTC-Source Ideographs and UAX #45

Unicode Standard Annex #45, "U-Source Ideographs" [UAX45] describes the format of USourceData.txt, which lists all of the information for UTC-Source ideographs.

3.8 Data File Comments

In addition to the specific documentation files for the UCD, individual data files often contain extensive header comments describing their content and any special conventions used in the data.

In some instances, individual property definition sections also contain comments with information about how the property may be derived. Such comments are informative; while they are intended to convey the intent of the derivation, in case of any mismatch between a statement of a derivation in a comment field and the actual listing of the derived property, the list is considered to be definitive. See Simple and Derived Properties.

3.9 Obsolete Documentation Files

UCD.html was formerly the primary documentation file for the UCD. As of Version 5.2.0, its content has been wholly incorporated into this document.

Unihan.html was formerly the primary documentation file for the Unihan Database. As of Version 5.1.0, its content has been wholly incorporated into [UAX38].

Versions of the Unicode Standard prior to Version 4.0.0 contained small, focused documentation files, UnicodeCharacterDatabase.html, PropList.html, and DerivedProperties.html, which were later consolidated into UCD.html.

StandardizedVariants.html has been obsoleted as of Version 9.0.0. See *Section 3.4, StandardizedVariants.html*.

4 UCD Files

The heart of the UCD consists of the data files themselves. This section describes the directory structure for the UCD, the format conventions for the data files, and provides documentation for data files not documented elsewhere in this annex.

4.1 Directory Structure

Each version of the UCD is released in a separate, numbered directory under the *Public* directory on the Unicode website. The content of that directory is complete for that release. It is also stable—once released, it will be archived permanently in that directory, unchanged, at a stable URL.

The specific files for the UCD associated with this version of the Unicode Standard (16.0.0) are located at:

https://www.unicode.org/Public/16.0.0/

The latest released version of the UCD is always accessible via the following stable URL:

https://www.unicode.org/Public/UCD/latest/

Zipped copies of the latest released version of the UCD are always accessible via the following stable URL:

https://www.unicode.org/Public/zipped/latest/

Prior to Version 6.3.0, access to the latest released version of the UCD was via the following stable URL:

https://www.unicode.org/Public/UNIDATA/

That "UNIDATA" URL will be maintained, but is no longer recommended, because it points to the *ucd* subdirectory of the latest release, rather than to the parent directory for the release. The "UNIDATA" naming convention is also very old, and does not follow the directory naming conventions currently used for other data releases in the *Public* directory on the Unicode website.

4.1.1 UCD Files Proper

The UCD proper is located in the *ucd* subdirectory of the numbered version directory. That directory contains all of the documentation files and most of the data files for the UCD, including some data files for derived properties.

Although all UCD data files are version-specific for a release and most contain internal date and version stamps, the file names of the released data files do not differ from version to version. When linking to a version-specific data file, the version will be indicated by the version number of the directory for the release.

All files for derived extracted properties are in the ***extracted*** subdirectory of the *ucd* subdirectory. See Derived Extracted Properties for documentation regarding those data files and their content.

A number of auxiliary properties are specified in files in the ***auxiliary*** subdirectory of the *ucd* subdirectory. It contains data files specifying properties associated with Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29] and with Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14], as well as test data for those algorithms. See Segmentation Test Files and Documentation for more information about the test data.

Certain data files associated with emoji properties are maintained in the ***emoji*** subdirectory of the *ucd* subdirectory. Those data files define the simple character properties associated with emoji characters, as well as the emoji variation sequences. Other data files associated with emoji, including those which define the RGI ("recommended for general interchange") sets of various types of emoji sequences, as well as emoji test data, are maintained elsewhere, and are not considered formally a part of the UCD. See [UTS51] for documentation regarding those data files and their content.

4.1.2 UCD XML Files

The XML version of the UCD is located in the *ucdxml* subdirectory of the numbered version directory. See the UCD in XML for more details.

4.1.3 Charts

The code charts specific to a version of Unicode are archived as a single large PDF file in the *charts* subdirectory of the numbered version directory. See the readme.txt in that subdirectory and the general web page explaining the Unicode Code Charts for more details.

4.1.4 Beta Review Considerations

Prior to the formal release of a version of the UCD, draft files are made available for review in a subdirectory named *draft*, under the /Public directory on the Unicode server. The files in this directory may include temporary files, including documentation of differences between draft versions. The number of reviews is not fixed—a beta review will always take place, but an alpha review is optional.

Notices contained in a ReadMe.txt file in the draft/UCD directory during the beta review period also make it clear that that directory contains preliminary material under review, rather than a final, stable release.

4.1.5 File Directory Differences for Early Releases

The UCD in XML was introduced in Version 5.1.0, so UCD directories prior to that do not contain the *ucdxml* subdirectory.

UCD directories prior to Version 13.0.0 do not contain the *emoji* subdirectory.

UCD directories prior to Version 4.1.0 do not contain the *auxiliary* subdirectory.

UCD directories prior to Version 3.2.0 do not contain the *extracted* subdirectory.

The general structure of the file directory for a released version of the UCD described above applies to Versions 4.1.0 and later. Prior to Version 4.1.0, versions of the UCD were not self-contained, complete sets of data files for that version, but instead only contained any new data files or any data files which had *changed* since the prior release.

Because of this, the property files for a given version prior to Version 4.1.0 can be spread over several directories. Consult the component listings at Enumerated Versions to find out which files in which directories comprise a complete set of data files for that version.

The directory naming conventions and the file naming conventions also differed prior to Version 4.1.0. So, for example, Version 4.0.0 of the UCD is contained in a directory named *4.0-Update*, and Version 4.0.1 of the UCD in a directory named *4.0-Update1*. Furthermore, for these earlier versions, the data file names *do* contain explicit version numbers.

4.2 File Format Conventions

Files in the UCD use the format conventions described in this section, unless otherwise specified.

4.2.1 Data Fields

- Each line of data consists of fields separated by semicolons. The fields are numbered starting with zero.
- The first field (0) of each line in the Unicode Character Database files represents a code point or range. The remaining fields (1..n) are properties associated with that code point.
- Leading and trailing spaces within a field are not significant. However, no leading or trailing spaces are allowed in any field of UnicodeData.txt. ~~For legacy reasons, no spaces are allowed before or after the semicolon in LineBreak.txt and in EastAsianWidth.txt.~~
- The Unihan data files [Unihan] in the UCD have a separate format, using tab characters instead of semicolons to separate fields. See [UAX38] for the detailed specification of the format of the Unihan data files. The data files TangutSources.txt and NushuSources.txt also use this format.

4.2.2 Code Points and Sequences

- Code points are expressed as hexadecimal numbers with four to six digits. (See *Appendix A, Notational Conventions* in [Unicode] for a full, formal definition of this convention.) They are written without the "U+" prefix in all data files except the Unihan data files. The Unihan data files use the "U+" prefix for all Unicode code points, to distinguish them from other decimal and hexadecimal numerical references occurring in their data fields.
- When a data field contains a sequence of code points, spaces separate the code points.

4.2.3 Code Point Ranges

- A range of code points is specified by the form "X..Y".
- Each code point in a range has the associated property value specified on a data file. For example (from Blocks.txt):

```
0000..007F; Basic Latin
0080..00FF; Latin-1 Supplement
```

- For backward compatibility, ranges in the file UnicodeData.txt are specified by entries for the start and end characters of the range, rather than by the form "X..Y". The start character is indicated by a range identifier, followed by a comma and the string "First", in angle brackets. This entry takes the place of a regular character name in field 1 for that line. The end character is indicated on the next line with the same range identifier, followed by a comma and the string "Last", in angle brackets:

```
4E00;<CJK Ideograph, First>;Lo;0;L;;;;;N;;;;;
9FEF;<CJK Ideograph, Last>;Lo;0;L;;;;;N;;;;;
```

  For character ranges using this convention, the names of all characters in the range are algorithmically derivable. See *Section 4.8, Name* in [Unicode] for more information on derivation of character names for such ranges.

4.2.4 Comments

- U+0023 NUMBER SIGN ("#") is used to indicate comments: all characters from the number sign to the end of the line are considered part of the comment, and are disregarded when parsing data.
- In many files, the comments on data lines use a common format, as illustrated here (from Scripts.txt):

```
09B2          ; Bengali # Lo       BENGALI LETTER LA
```

- The first part of a comment using this common format is the General_Category value, provided for information. This is followed by the character name for the code point in the first field (0).
- The printing of the General_Category value is suppressed in instances where it would be redundant, as for DerivedGeneralCategory.txt, in which the value of the property value in the data field is already the General_Category value.
- The symbol "L&" indicates characters of General_Category Lu, Ll, or Lt (uppercase, lowercase, or titlecase letter). For example:

```
0386          ; Greek # L&       GREEK CAPITAL LETTER ALPHA WITH TONOS
```

  L& as used in these comments is an alias for the derived LC value (cased letter) for the General_Category property, as documented in PropertyValueAliases.txt.

- When the data line contains a range of code points, this common format for a comment also indicates a range of character names, separated by "..", as illustrated here (from DerivedNumericType.txt):

```
00BC..00BE    ; Numeric # No   [3] VULGAR FRACTION ONE QUARTER..VULGAR FRACTION THREE QUARTERS
```

- Normally, consecutive characters with the same property value would be represented by a single code point range. In data files using this comment convention, such ranges are subdivided so that all characters in a range also have the same General_Category value (or LC). While this convention results in more ranges than are strictly necessary, it makes the contents of the ranges clearer.
- When a code point range occurs, the number of items in the range is included in the comment (in square brackets), immediately following the General_Category value.
- The comments are purely informational, and may change format or be omitted in the future. They should not be parsed for content. However, see Section 4.2.10 @missing Conventions.

4.2.5 Code Point Labels

- Surrogate code points, private-use characters, control codes, noncharacters, and unassigned code points have no names. When such code points are listed in the data files, for example to list their General_Category values, the comments use code point labels instead of character names. For example (from DerivedCoreProperties.txt):

```
2065          ; Default_Ignorable_Code_Point # Cn       <reserved-2065>
```

- Although code point labels are not formally character names and are not considered values of the Name property for characters, they are designed to be maintained as unique values within the namespace for Unicode character names. Hence, implementations can safely use them as identifiers for code points without overlap with actual character names.
- Code point labels use one of the tags as documented in *Section 4.8, Name* in [Unicode] and as shown in *Table 3*, followed by "-" and the code point expressed in hexadecimal. The entire label is then enclosed in angle brackets when listed in data files of the UCD.

Table 3. Code Point Label Tags

Tag	General_Category	Note
reserved	Cn	Noncharacter_Code_Point=F
noncharacter	Cn	Noncharacter_Code_Point=T
control	Cc	
private-use	Co	
surrogate	Cs	

4.2.6 Multiple Properties in One Data File

- When a file contains the specification for multiple properties, the second field specifies the name of the property and the third field specifies the property value. For example (from DerivedNormalizationProps.txt):

```
03D2  ; FC_NFKC; 03C5          # L&  GREEK UPSILON WITH HOOK SYMBOL
03D3  ; FC_NFKC; 03CD          # L&  GREEK UPSILON WITH ACUTE AND HOOK SYMBOL
```

4.2.7 Binary Property Values

- For binary properties, the second field specifies the name of the applicable property, with the implied value of the property being "True". Only the ranges of characters with the binary property value of "Y" (= True) are listed. For example (from PropList.txt):

```
1680       ; White_Space # Zs      OGHAM SPACE MARK
2000..200A ; White_Space # Zs [11] EN QUAD..HAIR SPACE
```

4.2.8 Multiple Values for Properties

- When a data file defines a property which may take multiple values for a single code point, the multiple values are expressed in a space-delimited list. For example (from ScriptExtensions.txt):

```
0640          ; Adlm Arab Mand Mani Phlp Rohg Sogd Syrc # Lm       ARABIC TATWEEL
```

- In some cases—but not all—the order of multiple elements in a space-delimited list may be significant. When the order of multiple elements is significant, it is documented along with the property itself. For example (from Unihan_Readings.txt), for the tag kMandarin, when there are two values for a code point, the first value is used to indicate a preferred pronunciation for zh-Hans (CN) and the second a preferred pronunciation for zh-Hant (TW).
- For further discussion, see Section 5.7.6 Properties Whose Values Are Sets of Values.

4.2.9 Default Values

- Entries for a code point may be omitted in a data file if the code point has a default value for the property in question.
- For most string-valued properties, including the definition of foldings and mappings, the default value is the code point of the character itself.

- For some string-valued properties which define a property that applies primarily to a small, defined set of code points, the default value is <none>, which is interpreted as no value is defined. (This contrasts with specification of an actual value consisting of an empty string. See Section 4.2.11 Empty Fields.) Current examples include Bidi_Paired_Bracket, as well as some Unihan-related properties.
- For miscellaneous properties which take strings as values, such as the Unicode Name property, the default value is an empty string.
- For binary properties except for Extended_Pictographic, the default value is always "N" (= False) and is always omitted.
- For enumerated and catalog properties, the default value is listed in a comment. For example (from Scripts.txt):

```
#  All code points not explicitly listed for Script
#  have the value Unknown (Zzzz).
```

- A few properties of the enumerated type have multiple default values. In those cases, comments in the file explain the code point ranges for applicable values. See also *Table 4*.
- Default values are also listed in specially-formatted comment lines, using the keyword "@missing". Parsers which extract and process these lines can algorithmically determine the default values for all code points. See @missing Conventions for details about the syntax and use of these lines.
- Because of the legacy format constraints for UnicodeData.txt, that file contains no specific information about default values for properties. The default values for fields in UnicodeData.txt are documented in *Table 4* below if they cannot be derived from the general rules about default values for properties.
- The file ArabicShaping.txt is also exceptional, because it omits the listing of many characters whose property value (jt=T) can be derived by rule. Adding an "@missing" line to that file would result in the wrong interpretation of Joining_Type values for omitted characters. The full explicit listing of Joining_Type values and the correct "@missing" line for the default Joining_Type value (jt=U) can be found in the file DerivedJoiningType.txt instead.

Default values for common catalog, enumeration, and numeric properties are listed in *Table 4*, along with the exceptional binary property, Extended_Pictographic. Further explanation is provided below the table, in those cases where the default values are complex, as indicated in the third column.

Table 4. Default Values for Properties

Property Name	Default Value(s)	Complex?
Age	Unassigned (= NA)	No
Bidi_Class	L, AL, R, BN, ET	Yes
Block	No_Block	No
Canonical_Combining_Class	Not_Reordered (= 0)	No
Decomposition_Type	None	No
East_Asian_Width	Neutral (= N), Wide (= W)	Yes
Extended_Pictographic	N (= False), Y (= True)	Yes
General_Category	Cn	No
Line_Break	Unknown (= XX), ID, PR	Yes
Numeric_Type	None	No
Numeric_Value	NaN	No
Script	Unknown (= Zzzz)	No
Vertical_Orientation	Rotated (= R), Upright (= U)	Yes

4.2.9.1 Complex Default Values

Complex default values are those which take multiple values, contingent on code point ranges or other conditions. Complex default values other than those specified in the "@missing" line are explicitly listed in the relevant property file, except for instances noted in this section. This means that a parser extracting property values from the UCD should never encounter an ambiguous condition for which the default value of a property for a particular code point is unclear.

- Bidi_Class:
  See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9] and DerivedBidiClass.txt for full details.
- East_Asian_Width:
  This property defaults to Neutral for most code points, but defaults to Wide for unassigned code points in blocks associated with CJK ideographs. See Unicode Standard Annex #11, "East Asian Width" [UAX11] and EastAsianWidth.txt for documentation of the default values and DerivedEastAsianWidth.txt for the full listing of values.
- Line_Break:
  This property defaults to Unknown for most code points, but defaults to ID for unassigned code points in blocks associated with CJK ideographs, and in blocks in the ranges U+1F000..U+1FAFF and U+1FC00..U+1FFFD. The property defaults to PR for unassigned code points in the Currency Symbols block. See Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14] and LineBreak.txt for documentation of the default values and DerivedLineBreak.txt for the full listing of values.
- Extended_Pictographic:
  This property defaults to N (= False) for most code points, but defaults to Y (= True) for unassigned code points in blocks in the ranges U+1F000..U+1FAFF and U+1FC00..U+1FFFD. Those ranges are correlated with the ranges associated with default values for the Line_Break property, and have the same rationale. They help future-proof the behavior of Unicode segmentation algorithms for code point ranges most likely to be used for future assignment of new emoji characters.
- Vertical_Orientation:
  This property defaults to Rotated (R) for most code points, but defaults to Upright (U) for unassigned code points in blocks associated with scripts that are themselves predominantly Upright, in blocks for some notational systems, and in blocks predominantly associated

with pictographic symbols and emoji. See Unicode Standard Annex #50, "Unicode Vertical Text Layout" [UAX50] and VerticalOrientation.txt for full details.

4.2.10 @missing Conventions

Specially-formatted comment lines with the keyword "@missing" are used to define default property values for ranges of code points not explicitly listed in a data file. These lines follow regular conventions that make them machine-readable.

An @missing line starts with the comment character "#", followed by a space, then the "@missing" keyword, followed by a colon, another space, a code point range, and a semicolon. Then the line typically continues with a semicolon-delimited list of one or more default property values. For example:

```
# @missing: 0000..10FFFF; Unknown
```

In general, the code point range and semicolon-delimited list follow the same syntactic conventions as the data file in which the @missing line occurs, so that any parser which interprets that data file can easily be adapted to also parse and interpret an @missing line to pick up default property values for code points.

@missing lines are also supplied for many properties in the file PropertyValueAliases.txt. In this case, because there are many @missing lines in that single data file, each @missing line in that file uses the syntactic pattern code_point_range; property_name; default_prop_val.

An @missing line is never provided for a binary property, because the default value for binary properties is always "N" and need not be defined redundantly for each binary property.

Because of the addition of property names when @missing lines are included in PropertyValueAliases.txt, there are currently two syntactic patterns used for @missing lines, as summarized schematically below:

1. code_point_range; default_prop_val
2. code_point_range; property_name; default_prop_val

In this schematic representation, "default_prop_val" stands in for either an explicit property value or for a special tag such as <none> or <script>.

Pattern #1 is used in most primary and derived UCD files. For example:

```
# @missing: 0000..10FFFF; <none>
```

Pattern #2 is used in PropertyValueAliases.txt and in DerivedNormalizationProps.txt, both of which contain values associated with many properties. For example:

```
# @missing: 0000..10FFFF; NFD_QC; Yes
```

The special tag values which may occur in the default_prop_val field in an @missing line are interpreted as follows:

Tag	Interpretation
<none>	no value is defined
<code point>	the string representation of the code point value
<script>	the value equal to the Script property value for this code point

Starting with Version 15.0, some data files in the UCD may contain multiple @missing lines defined for the *same* property. When multiple @missing lines are defined this way, they are to be interpreted as follows: Each successive @missing line specifies an *overriding* range value for all previous @missing definitions. This convention allows a generic default value to be specified first for the entire Unicode code point range, followed by other specific default values for more constrained, specific sub-ranges. This enables an easy-to-understand and easy-to-maintain way of handling complex default values, as for the Bidi_Class or Line_Break properties. (See Complex Default Values.) The following simple example for East_Asian_Width, extracted from DerivedEastAsianWidth.txt, illustrates this mechanism:

```
# @missing: 0000..10FFFF; Neutral
# @missing: 3400..4DBF; Wide
# @missing: 4E00..9FFF; Wide
# @missing: F900..FAFF; Wide
# @missing: 20000..2FFFD; Wide
# @missing: 30000..3FFFD; Wide
```

Implementation of parsing for multiple @missing lines for a single property is straightforward. Each time an @missing line is encountered, simply assign the given default value to the specified range. With this strategy, each successive @missing line will automatically override any prior assigned values for a given sub-range.

4.2.11 Empty Fields

The data file UnicodeData.txt defines many property values in each record. When a field in a data line for a code point is empty, that indicates that the property takes the default value for that code point. For example:

```
0022;QUOTATION MARK;Po;0;ON;;;;;N;;;;;
```

In that data line, the empty numeric fields indicate that the value of Numeric_Value for U+0022 is NaN and that the value of Numeric_Type is None. The empty case mapping fields indicate that the value of Simple_Uppercase_Mapping for U+0022 takes the default value, namely the code point itself, and so forth.

The interpretation of empty fields in other data files of the UCD differs. In the case of data files which define string-valued properties, the omission of an entry for a code point indicates that the property takes the default value for that code point. However, if there is an entry for a code point, but the property value field for that entry is empty, that indicates that the property value is an explicit empty string (""). For example, the derived property NFKC_Casefold may map a code point to a sequence of code points, to a single different code point, to the same single code point, or to no code point at all (an empty string). See the following entries from the data file DerivedNormalizationProps.txt:

```
00AA          ; NFKC_CF; 0061           # Lo       FEMININE ORDINAL INDICATOR
00AD          ; NFKC_CF;                # Cf       SOFT HYPHEN
00AF          ; NFKC_CF; 0020 0304      # Sk       MACRON
```

The empty field for U+00AD indicates that the property NFKC_Casefold maps SOFT HYPHEN to an empty string. By contrast, the absence of the entry for U+00AE in the data file indicates that the property NFKC_Casefold maps U+00AE REGISTERED SIGN to itself—the default value.

4.2.12 Text Encoding

- The data files use UTF-8. Unless otherwise noted, non-ASCII characters only appear in comments.
- The Unihan data files [Unihan] in the UCD make extensive use of UTF-8 in data fields. (See [UAX38] for details.)
- For legacy reasons, NamesList.txt was exceptional; it was encoded in Latin-1 prior to Unicode 6.2. For Unicode 6.2 and later, the encoding is UTF-8. See NamesList.html.
- Segmentation test data files, such as WordBreakTest.txt, make use of non-ASCII (UTF-8) characters as delimiters for data fields.

4.2.13 Line Termination

- All data files in the UCD use LF line termination (not CRLF line termination). When copied to different systems, these line endings may be automatically changed to use the native line termination conventions for that system. Make sure your editor (or parser) can deal with the line termination style in the local copy of the data files.

4.2.14 Other Conventions

- In some test data files, segments of the test data are distinguished by a line starting with an "@" sign. For example (from NormalizationTest.txt):

```
@Part1 # Character by character test
```

4.2.15 Other File Formats

- The data format for Unihan data files and for TangutSources.txt and NushuSources.txt in the UCD differs from the standard format. See the discussion of Unihan and UAX #38 earlier in this annex for more information.
- The format for NamesList.txt, which documents the Unicode names list and which is used programmatically to drive the formatting program for Unicode code charts, also differs significantly from regular UCD data files. See NamesList.html
- Index.txt is another exception. It uses a tab-delimited format, with field 0 consisting of an index entry string, and field 1 a code point. Index.txt is used to maintain the Unicode Character Name Index.
- The various segmentation test data files make use of "#" to delimit comments, but have distinct conventions for their data fields. See the documentation in their header sections for details of the data field formats for those files.
- The XML version of the UCD has its own file format conventions. In those files, "#" is used to stand for the code point in algorithmically derivable character names such as CJK UNIFIED IDEOGRAPH-4E00 or TANGUT IDEOGRAPH-17000, so as to allow for name sharing in more compact representations of the data. See Unicode Standard Annex #42, "Unicode Character Database in XML" [UAX42] for details.

4.3 File List

The exact list of files associated with any particular version of the UCD is available on the Unicode website by referring to the component listings at Enumerated Versions.

The majority of the data files in the UCD provide specifications of character properties for Unicode characters. Those files and their contents are documented in detail in the Property Definitions section below.

The data files in the *extracted* subdirectory constitute reformatted listings of single character properties extracted from UnicodeData.txt or other primary data files. The reformatting is provided to make it easier to see the particular set of characters having certain values for enumerated properties, or to separate the statement of that property from other properties defined together in UnicodeData.txt. These files also include explicit listings of default values for the respective properties. These extracted, derived data files are further documented in the Derived Extracted Properties section below.

The UCD also contains a number of test data files, whose purpose is to provide standard test cases useful in verifying the implementation of complex Unicode algorithms. See the Test Files section below for more documentation.

The remaining files in the Unicode Character Database do not directly specify Unicode properties. The important ones and their functions are listed in *Table 5*. The Status column indicates whether the file (and its content) is considered **N**ormative, **I**nformative, or **P**rovisional.

Table 5. UCD Files That Do Not Specify Character Properties

File Name	Reference	Status	Description

CJKRadicals.txt	[UAX38]	I	List of Unified CJK Ideographs and CJK Radicals that correspond to specific radical numbers used in the CJK radical stroke counts.
USourceData.txt	[UAX45]	N	The list of formal references for UTC-Source ideographs, together with data regarding their status and sources.
USourceGlyphs.pdf	[UAX45]	I	A table containing a representative glyph for each UTC-Source ideograph.
USourceRSChart.pdf	[UAX45]	I	A radical-stroke index of all the UTC-Source ideographs.
TangutSources.txt	Chapter 18	N	Specifies normative source mappings for Tangut ideographs and components. This data file also includes informative radical-stroke values that are used in the preparation of the code charts for the Tangut blocks. **kTGT_MergedSrc**: normative source mapping to various Tangut source references **kRSTUnicode**: informative radical-stroke value
NushuSources.txt	Chapter 18	N	Specifies normative source mappings for Nushu ideographs. This data file also includes informative readings for Nushu characters. **kSrc_NushuDuben**: normative source mapping to the Nushu Duben **kReading**: informative example phonetic reading
EmojiSources.txt	Chapter 22	N	Specifies source mappings to SJIS values for emoji symbols in the original implementations of these symbols by Japanese telecommunications companies.
Index.txt	Chapter 24	I	Index to Unicode characters.
NamesList.txt	Chapter 24	I	Names list used for production of the code charts, derived from UnicodeData.txt. It contains additional annotations.
NamesList.html	Chapter 24	I	Documents the format of NamesList.txt.
StandardizedVariants.txt	Chapter 23	N	Lists all the standardized variant sequences that have been defined, plus a textual description of their desired appearance.
StandardizedVariants.html	Chapter 23	N	An obsolete derived documentation file.
NamedSequences.txt	[UAX34]	N	Lists the names for all approved named sequences. This is a string-valued property of strings.
NamedSequencesProv.txt	[UAX34]	P	Lists the names for all provisional named sequences. This is a (provisional) string-valued property of strings.
emoji-variation-sequences.txt	[UTS51]	N	Lists all emoji presentation sequences and text presentation sequences involving currently encoded emoji characters.

For more information about these files and their use, see the referenced annexes or chapters of Unicode Standard, or, in the case of emoji sequences data, [UTS51].

4.4 Zipped Files

Starting with Version 4.1.0, zipped versions of all of the UCD files, both data files and documentation files, are available under the *Public/zipped* directory on the Unicode website. Each collection of zipped files is located there in a numbered subdirectory corresponding to that version of the UCD.

Two different zipped files are provided for each version:

- **Unihan.zip** is the zipped version of the very large Unihan data files
- **UCD.zip** is the zipped version of all of the rest of the UCD data files, excluding the Unihan data files.

This bifurcation allows for better management of downloading version-specific information, because Unihan.zip contains all the pertinent CJK-related property information, while UCD.zip contains all of the rest of the UCD property information, for those who may not need the voluminous CJK data.

Starting with Version 6.1.0 the main versioned directories for the UCD also contain a copy of UCD.zip, for convenience in access.

In versions of the UCD prior to Version 4.1.0, zipped copies of the Unihan data files (which for those versions were released as a single large text file, Unihan.txt) are provided in the same directory as the UCD data files. These zipped files are only posted for versions of the UCD in which Unihan.txt was updated.

4.5 UCD in XML

Starting with Version 5.1.0, a set of XML data files are also released with each version of the UCD. Those data files make it possible to import and process the UCD property data using standard XML parsing tools, instead of the specialized parsing required for the various individual data files of the UCD.

4.5.1 UAX #42

Unicode Standard Annex #42, "Unicode Character Database in XML" [UAX42] defines an XML schema which is used to incorporate all of the Unicode character property information into the XML version of the UCD. See that annex for details of the schema and conventions regarding the grouping of property values for more compact representations.

4.5.2 XML File List

The XML version of the UCD is contained in the *ucdxml* subdirectory of the UCD. The files are all zipped. The list of files is shown in *Table 6*.

Table 6. XML File List

File Name	CJK	non-CJK
ucd.all.flat.zip	x	x
ucd.all.grouped.zip	x	x
ucd.nounihan.flat.zip		x
ucd.nounihan.grouped.zip		x
ucd.unihan.flat.zip	x	
ucd.unihan.grouped.zip	x	

The "flat" file versions simply list all attributes with no particular compression. The "grouped" file versions apply the grouping mechanism described in [UAX42] to cut down on the size of the data files.

5 Properties

This section documents the Unicode character properties, relating them in detail to the particular UCD data files in which they are specified. For enumerated properties in particular, this section also documents the actual values which those properties can have.

5.1 Property Index

Table 7 provides a summary list of the Unicode character properties, excluding most of those specific to the Unihan data files [Unihan]. For a comparable index of CJK character properties, see Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38].

The properties are roughly organized into groups based on their usage. This grouping is primarily for documentation convenience and except for contributory properties, has no normative implications. Contributory properties are shown in this index with a gray background, to better distinguish them visually from ordinary (simple or derived) properties. Deprecated properties and other properties not recommended for support in public property APIs are also shown with a gray background. The link on each property leads to its description in *Table 9, Property Table*. Any property marked as deprecated in this index is also automatically considered obsolete.

Table 7. Property Index by Scope of Use

General	Numeric	Segmentation
Name	Numeric_Value	Line_Break
Name_Alias	Numeric_Type	Grapheme_Cluster_Break
Block	Hex_Digit	Sentence_Break
Age	ASCII_Hex_Digit	Word_Break
General_Category	**Normalization**	**CJK**
Script	Canonical_Combining_Class	Ideographic
Script_Extensions	Decomposition_Mapping	Unified_Ideograph
White_Space	Composition_Exclusion	Radical
Alphabetic	Full_Composition_Exclusion	IDS_Unary_Operator
Hangul_Syllable_Type	Decomposition_Type	IDS_Binary_Operator
Noncharacter_Code_Point	FC_NFKC_Closure (deprecated)	IDS_Trinary_Operator
Default_Ignorable_Code_Point	NFC_Quick_Check	Unicode_Radical_Stroke
Deprecated	NFKC_Quick_Check	Equivalent_Unified_Ideograph
Logical_Order_Exception	NFD_Quick_Check	**Miscellaneous**
Variation_Selector	NFKD_Quick_Check	Math
Case	Expands_On_NFC (deprecated)	Quotation_Mark
Uppercase	Expands_On_NFD (deprecated)	Dash
Lowercase	Expands_On_NFKC (deprecated)	Hyphen (deprecated, stabilized)
Lowercase_Mapping	Expands_On_NFKD (deprecated)	Sentence_Terminal
Titlecase_Mapping	NFKC_Casefold	Terminal_Punctuation
Uppercase_Mapping	Changes_When_NFKC_Casefolded	Diacritic
Case_Folding	NFKC_Simple_Casefold	Extender
Simple_Lowercase_Mapping	**Shaping and Rendering**	Grapheme_Base
Simple_Titlecase_Mapping	Join_Control	Grapheme_Extend
Simple_Uppercase_Mapping	Joining_Group	Grapheme_Link (deprecated)
Simple_Case_Folding	Joining_Type	Unicode_1_Name
Soft_Dotted	Modifier_Combining_Mark	ISO_Comment (deprecated, stabilized)
Cased	Vertical_Orientation	Regional_Indicator

Case_Ignorable	East_Asian_Width	Indic_Conjunct_Break
Changes_When_Lowercased	Prepended_Concatenation_Mark	Indic_Positional_Category
Changes_When_Uppercased	**Bidirectional**	Indic_Syllabic_Category
Changes_When_Titlecased	Bidi_Class	**Contributory Properties**
Changes_When_Casefolded	Bidi_Control	Other_Alphabetic
Changes_When_Casemapped	Bidi_Mirrored	Other_Default_Ignorable_Code_Point
Emoji	Bidi_Mirroring_Glyph	Other_Grapheme_Extend
Emoji	Bidi_Paired_Bracket	Other_ID_Start
Emoji_Presentation	Bidi_Paired_Bracket_Type	Other_ID_Continue
Emoji_Modifier	**Identifiers**	Other_Lowercase
Emoji_Modifier_Base	ID_Continue	Other_Math
Emoji_Component	ID_Start	Other_Uppercase
Extended_Pictographic	XID_Continue	Jamo_Short_Name
	XID_Start	
	ID_Compat_Math_Continue	
	ID_Compat_Math_Start	
	Pattern_Syntax	
	Pattern_White_Space	

5.2 About the Property Table

Table 9, Property Table specifies the list of character properties defined in the UCD. That table is divided into separate sections for each data file in the UCD. Data files which define a single property or a small number of properties are listed first, followed by the data files which define a large number of properties: DerivedCoreProperties.txt, DerivedNormalizationProps.txt, PropList.txt, UnicodeData.txt, and emoji-data.txt. In some instances for these files defining many properties, the entries in the property table are grouped by type, for clarity in presentation, rather than being listed alphabetically.

In *Table 9, Property Table* each property is described as follows:

First Column. This column contains the name of each of the character properties specified in the respective data file. Any special status for a property, such as whether it is obsolete, deprecated, or stabilized, is also indicated in the first column.

Second Column. This column indicates the type of the property, according to the key in *Table 8*.

Table 8. Property Type Key

Property Type	Symbol	Examples
Catalog	C	Age, Block
Enumeration	E	Joining_Type, Line_Break
Binary	B	Uppercase, White_Space
String-valued	S	Uppercase_Mapping, Case_Folding
Numeric	N	Numeric_Value
Miscellaneous	M	Name, Jamo_Short_Name

- **Catalog** properties have enumerated values which are expected to be regularly extended in successive versions of the Unicode Standard. This distinguishes them from Enumeration properties.
- **Enumeration** properties have enumerated values which constitute a logical partition space; new values will generally *not* be added to them in successive versions of the standard.
- **Binary** properties are a special case of Enumeration properties, which have exactly two values: Yes and No (or True and False).
- **String-valued** properties are typically mappings from a Unicode code point to another Unicode code point or sequence of Unicode code points; examples include case mappings and decomposition mappings.
- Properties of strings are properties defined for strings; in other words, their domain is a set of strings rather than a set of characters or code points. Properties of strings are sometimes called "string properties" for short. For example, the file NamedSequences.txt defines names (which are themselves string values) for a certain set of specific character sequences. Properties of strings are not explicitly listed for the UCD in the Property Table, and hence are given no specific type symbol in the Property Type Key.
- **Numeric** properties specify the actual numeric values for digits and other characters associated with numbers in some way.
- **Miscellaneous** properties are those properties that do not fit neatly into the other property categories; they currently include character names, comments about characters, the Script_Extensions property, and the Unicode_Radical_Stroke property (a combination of numeric values) documented in Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38].

For a more complete discussion of types of character properties, including formal definitions, see Unicode Technical Report 23, "The Unicode Character Property Model" [UTR23].

Third Column. This column indicates the status of the property: **N**ormative or **I**nformative or **C**ontributory or **P**rovisional.

Fourth Column. This column provides a description of the property or properties. This includes information on derivation for derived properties, as well as references to locations in the standard where the property is defined or discussed in detail.

In the section of the table for UnicodeData.txt, the data field numbers are also supplied in parentheses at the start of the description.

For a few entries in the property table, values specified in the fields in a data file only contribute to a full definition of a Unicode character property. For example, the values in field 1 (Name) in UnicodeData.txt do not provide all the values for the Name property for all code points; Jamo.txt must also be used, and the Name property for CJK unified ideographs, Tangut ideographs, Khitan Small Script ideographs, and Nushu ideographs is derived by rule.

None of the Unicode character properties should be used simply on the basis of the descriptions in the property table without consulting the relevant discussions in the Unicode Standard. Because of the enormous variety of characters in the repertoire of the Unicode Standard, character properties tend not to be self-evident in application, even when the names of the properties may seem familiar from their usage with much smaller legacy character encodings.

5.3 Property Definitions

This section contains the table which describes each character property and defines its status, organized by data file in the UCD. *Table 9* provides general descriptions of the Unicode character properties, their derivations, and/or their usage, as well as pointers to the respective parts of the standard where formal property definitions or additional information about the properties can be found. The property status column and any formal statement of the derivation of derived properties are definitive; however, *Table 9* does not provide formal definitions of the other properties and should not be interpreted as such. For details on the columns and overall organization of the table, see Section 5.2 About the Property Table.

Table 9. Property Table

ArabicShaping.txt			
Joining_Type Joining_Group	E	N	Basic Arabic and Syriac character shaping properties, such as initial, medial and final shapes. See *Section 9.2, Arabic* in [Unicode].
BidiBrackets.txt			
Bidi_Paired_Bracket_Type	E	N	Type of a paired bracket, either opening or closing. This property is used in the implementation of parenthesis matching. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9].
Bidi_Paired_Bracket	S	N	For an opening bracket, the code point of the matching closing bracket. For a closing bracket, the code point of the matching opening bracket. This property is used in the implementation of parenthesis matching. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9].
BidiMirroring.txt			
Bidi_Mirroring_Glyph	S	I	Informative mapping for substituting characters in an implementation of bidirectional mirroring. This maps a subset of characters with the Bidi_Mirrored property to other characters that normally are displayed with the corresponding mirrored glyph. When a character with the Bidi_Mirrored property has the default value for Bidi_Mirroring_Glyph, that means that no other character exists whose glyph is appropriate for character-based glyph mirroring. Implementations must then use other mechanisms to implement mirroring of those characters for the Unicode Bidirectional Algorithm. See Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9]. Do not confuse this property with the Bidi_Mirrored property itself.
Blocks.txt			
Block	C	N	Blocks.txt specifies the Block property, which consists of the list of block names for ranges of code points. See D10b in *Section 3.4, Characters and Encoding*, of [Unicode]. See also the code charts in [Unicode].
CompositionExclusions.txt			
Composition_Exclusion	B	N	A property used in normalization. See Unicode Standard Annex #15, "Unicode Normalization Forms" [UAX15]. Unlike other files, CompositionExclusions.txt simply lists the relevant code points.
CaseFolding.txt			
Simple_Case_Folding Case_Folding	S	N	Mapping from characters to their case-folded forms. This is an informative file containing normative derived properties. *Derived from UnicodeData and SpecialCasing.* **Note:** The case foldings are omitted in the data file if they are the same as the code point itself.
DerivedAge.txt			
Age	C	N	A property defining when various code points were designated/assigned in successive versions of the Unicode Standard. For a detailed discussion of the Age property, see Section 5.14, *Character Age*.
EastAsianWidth.txt			

East_Asian_Width	E	N	A property for determining the choice of wide versus narrow glyphs in East Asian contexts. Property values are described in Unicode Standard Annex #11, "East Asian Width" [UAX11]. **Note:** Some values of the East_Asian_Width property are used in the derivation of Line_Break property values, and hence are pertinent to line breaking behavior. See Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14].
EquivalentUnifiedIdeograph.txt			
Equivalent_Unified_Ideograph	S	I	A property which maps most CJK radicals and CJK strokes to the most reasonably equivalent CJK unified ideograph.
HangulSyllableType.txt			
Hangul_Syllable_Type	E	N	The values L, V, T, LV, and LVT used in *Chapter 3, Conformance* in [Unicode].
IndicPositionalCategory.txt			
Indic_Positional_Category	E	I	A property informally defining the positional categories for dependent vowels, viramas, combining marks, and other characters used in Indic scripts. General descriptions of the property values are provided in the header section of the data file IndicPositionalCategory.txt.
IndicSyllabicCategory.txt			
Indic_Syllabic_Category	E	I	A property informally defining the structural categories of syllabic components in Indic scripts. General descriptions of the property values are provided in the header section of the data file IndicSyllabicCategory.txt.
Jamo.txt			
Jamo_Short_Name	M	C	The Hangul Syllable names are derived from the Jamo Short Names, as described in *Chapter 3, Conformance* in [Unicode].
LineBreak.txt			
Line_Break	E	N	A property for line breaking. For more information, see Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14].
GraphemeBreakProperty.txt			
Grapheme_Cluster_Break	E	I	See Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29]
SentenceBreakProperty.txt			
Sentence_Break	E	I	See Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29]
WordBreakProperty.txt			
Word_Break	E	I	See Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29]
NameAliases.txt			
Name_Alias	M	N	Normative formal aliases for characters with erroneous names, for control characters and some format characters, and for character abbreviations, as described in *Chapter 4, Character Properties* in [Unicode]. Aliases tagged with the type "correction", as well as a selection of aliases of other types, are published in the Unicode Standard code charts.
NormalizationCorrections.txt			
used in Decomposition Mappings	S	N	NormalizationCorrections lists code point differences for *Normalization Corrigenda*. For more information, see Unicode Standard Annex #15, "Unicode Normalization Forms" [UAX15].
Scripts.txt			
Script	C	I	Script values for use in regular expressions and elsewhere. For more information, see Unicode Standard Annex #24, "Unicode Script Property" [UAX24].
ScriptExtensions.txt			
Script_Extensions	M	I	Enumerated sets of Script values for use in regular expressions and elsewhere. For more information, see Unicode Standard Annex #24, "Unicode Script Property" [UAX24].
SpecialCasing.txt			
Uppercase_Mapping Lowercase_Mapping Titlecase_Mapping	S	I	Data for producing (in combination with the simple case mappings from UnicodeData.txt) the full case mappings.
Unihan data files [Unihan] (for more information, see [UAX38])			
Numeric_Type Numeric_Value	E N	I	The characters tagged in the Unihan data files with either kPrimaryNumeric, kAccountingNumeric, or kOtherNumeric are given the property value Numeric_Type=Numeric, and their Numeric_Value is set to the first value indicated in those tags. (These three tags occasionally contain comma-separated multiple values, which is why the Numeric_Value is specified as the *first* of those values in the data file. The three tags, kPrimaryNumeric, kAccountingNumeric, and kOtherNumeric are mutually exclusive, so no character has more than one of those tags.)

			Most characters have these numeric properties based on values from UnicodeData.txt. See Numeric_Type.
Unicode_Radical_Stroke	M	I	The Unicode radical-stroke count, based on the tag kRSUnicode.
VerticalOrientation.txt			
Vertical_Orientation	E	I	A property used to establish a default for the correct orientation of characters when used in vertical text layout, as described in Unicode Standard Annex #50, "Unicode Vertical Text Layout" [UAX50].
DerivedCoreProperties.txt			
Lowercase	B	I	Characters with the Lowercase property. For more information, see *Chapter 4, Character Properties* in [Unicode]. *Generated from: Ll + Other_Lowercase*
Uppercase	B	I	Characters with the Uppercase property. For more information, see *Chapter 4, Character Properties* in [Unicode]. *Generated from: Lu + Other_Uppercase*
Cased	B	I	Characters which are considered to be either uppercase, lowercase or titlecase characters. This property is not identical to the Changes_When_Casemapped property. For more information, see D135 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: Lowercase + Uppercase + Lt*
Case_Ignorable	B	I	Characters which are ignored for casing purposes. For more information, see D136 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: Mn + Me + Cf + Lm + Sk + Word_Break=MidLetter + Word_Break=MidNumLet + Word_Break=Single_Quote*
Changes_When_Lowercased	B	I	Characters whose normalized forms are not stable under a toLowercase mapping. For more information, see D139 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: toLowercase(toNFD(X)) != toNFD(X)*
Changes_When_Uppercased	B	I	Characters whose normalized forms are not stable under a toUppercase mapping. For more information, see D140 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: toUppercase(toNFD(X)) != toNFD(X)*
Changes_When_Titlecased	B	I	Characters whose normalized forms are not stable under a toTitlecase mapping. For more information, see D141 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: toTitlecase(toNFD(X)) != toNFD(X)*
Changes_When_Casefolded	B	I	Characters whose normalized forms are not stable under case folding. For more information, see D142 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: toCasefold(toNFD(X)) != toNFD(X)*
Changes_When_Casemapped	B	I	Characters which may change when they undergo case mapping. For more information, see D143 in *Section 3.13, Default Case Algorithms* in [Unicode]. *Generated from: Changes_When_Lowercased(X) or Changes_When_Uppercased(X) or Changes_When_Titlecased(X)*
Alphabetic	B	I	Characters with the Alphabetic property. For more information, see *Chapter 4, Character Properties* in [Unicode]. *Generated from: Lowercase + Uppercase + Lt + Lm + Lo + Nl + Other_Alphabetic*
Default_Ignorable_Code_Point	B	N	For programmatic determination of default ignorable code points. New characters that should be ignored in rendering (unless explicitly supported) will be assigned in these ranges, permitting programs to correctly handle the default rendering of such characters when not otherwise supported. For more information, see the FAQ Display of Unsupported Characters, and *Section 5.21, Ignoring Characters in Processing* in [Unicode].

			Generated from: *Other_Default_Ignorable_Code_Point* *+ Cf (Format characters)* *+ Variation_Selector* *- White_Space* *- FFF9..FFFB (Interlinear annotation format characters)* *- 13430..1343F (Egyptian hieroglyph format characters)* *- Prepended_Concatenation_Mark (Exceptional format characters that should be visible)*
Grapheme_Base	B	N	Property used together with the definition of Standard Korean Syllable Block to define "Grapheme base". See D58 in *Chapter 3, Conformance* in [Unicode]. *Generated from: [0..10FFFF] - Cc - Cf - Cs - Co - Cn - Zl - Zp - Grapheme_Extend* **Note:** Grapheme_Base is a property of individual characters. That usage contrasts with "grapheme base", which is an attribute of Unicode strings; a grapheme base may consist of a Korean syllable which is itself represented by a sequence of conjoining jamos.
Grapheme_Extend	B	N	Property used to define "Grapheme extender". See D59 in *Chapter 3, Conformance* in [Unicode]. *Generated from: Me + Mn + Other_Grapheme_Extend* **Note:** The set of characters for which Grapheme_Extend=Yes is used in the derivation of the property value Grapheme_Cluster_Break=Extend. Grapheme_Cluster_Break=Extend consists of the set of characters for which Grapheme_Extend=Yes *or* Emoji_Modifier=Yes. See [UAX29] and [UTS51].
Grapheme_Link (Deprecated as of 5.0.0)	B	I	Formerly proposed for programmatic determination of grapheme cluster boundaries. *Generated from: Canonical_Combining_Class=Virama*
Indic_Conjunct_Break	E	I	This property defines values used in Grapheme Cluster Break algorithm in [UAX29]. *Generated as follows:* • Define the set of applicable scripts. For Unicode 15.1, the set is defined as S = [\p{sc=Beng}\p{sc=Deva}\p{sc=Gujr}\p{sc=Mlym}\p{sc=Orya}\p{sc=Telu}] • Then for any character C: 1. InCB = Linker iff C in [S &\p{Indic_Syllabic_Category=Virama}] 2. InCB = Consonant iff C in [S &\p{Indic_Syllabic_Category=Consonant}] 3. InCB = Extend iff C in [[\p{gcb=Extend}-\p{ccc=0}] \p{gcb=ZWJ} -\p{Indic_Syllabic_Category=Virama} -\p{Indic_Syllabic_Category=Consonant}] 4. Otherwise, InCB = None (the default value)
Math	B	I	Characters with the Math property. For more information, see *Chapter 4, Character Properties* in [Unicode]. *Generated from: Sm + Other_Math*
ID_Start	B	I	Used to determine programming identifiers, as described in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax" [UAX31].
ID_Continue	B	I	
XID_Start	B	I	
XID_Continue	B	I	
DerivedNormalizationProps.txt			
Full_Composition_Exclusion	B	N	Characters that are excluded from composition: those listed explicitly in CompositionExclusions.txt, plus the derivable sets of *Singleton Decompositions* and *Non-Starter Decompositions*, as documented in that data file.
Expands_On_NFC Expands_On_NFD Expands_On_NFKC Expands_On_NFKD (Deprecated as of 6.0.0)	B	N	Characters that expand to more than one character in the specified normalization form.
FC_NFKC_Closure (Deprecated as of 6.0.0)	S	N	Characters that require extra mappings for closure under Case Folding plus Normalization Form KC. The mapping is listed in Field 2.
NFD_Quick_Check NFKD_Quick_Check	E	N	For property values, see Decompositions and Normalization. (Abbreviated names: NFD_QC, NFKD_QC, NFC_QC, NFKC_QC)

NFC_Quick_Check NFKC_Quick_Check			
NFKC_Casefold	S	I	A mapping designed for best behavior when doing caseless matching of strings interpreted as identifiers. (Abbreviated name: NFKC_CF) For the definition of the related string transform toNFKC_Casefold() based on this mapping, see *Section 3.13, Default Case Algorithms* in [Unicode]. The mapping is listed in Field 2.
Changes_When_NFKC_Casefolded	B	I	Characters which are not identical to their NFKC_Casefold mapping. *Generated from: (cp != NFKC_CaseFold(cp))*
NFKC_Simple_Casefold	S	I	A mapping designed for best behavior when doing simple caseless matching of strings interpreted as identifiers. (Abbreviated name: NFKC_SCF) The mapping is listed in Field 2.
PropList.txt			
ASCII_Hex_Digit	B	N	ASCII characters commonly used for the representation of hexadecimal numbers.
Bidi_Control	B	N	Format control characters which have specific functions in the Unicode Bidirectional Algorithm [UAX9].
Dash	B	I	Punctuation characters explicitly called out as dashes in the Unicode Standard, plus their compatibility equivalents. Most of these have the General_Category value Pd, but some have the General_Category value Sm because of their use in mathematics.
Deprecated	B	N	For a machine-readable list of deprecated characters. No characters will ever be removed from the standard, but the usage of deprecated characters is strongly discouraged.
Diacritic	B	I	Characters that linguistically modify the meaning of another character to which they apply. Some diacritics are not combining characters, and some combining characters are not diacritics.
Extender	B	I	Characters whose principal function is to extend the value of a preceding alphabetic character or to extend the shape of adjacent characters. Typical of these are length marks, iteration marks, and the Arabic *tatweel*.
Hex_Digit	B	I	Characters commonly used for the representation of hexadecimal numbers, plus their compatibility equivalents.
Hyphen (Stabilized as of 4.0.0; Deprecated as of 6.0.0)	B	I	Dashes which are used to mark connections between pieces of words, plus the *Katakana middle dot*. The *Katakana middle dot* functions like a hyphen, but is shaped like a dot rather than a dash.
Ideographic	B	I	Characters considered to be CJKV (Chinese, Japanese, Korean, and Vietnamese) or other siniform (Chinese writing-related) ideographs. This property roughly defines the class of "Chinese characters" and does not include characters of other logographic scripts such as Cuneiform or Egyptian Hieroglyphs. The Ideographic property is used in the definition of Ideographic Description Sequences.
ID_Compat_Math_Start	B	I	Used in mathematical identifier profile in UAX #31.
ID_Compat_Math_Continue	B	I	Used in mathematical identifier profile in UAX #31.
IDS_Unary_Operator	B	N	Used in Ideographic Description Sequences.
IDS_Binary_Operator	B	N	Used in Ideographic Description Sequences.
IDS_Trinary_Operator	B	N	Used in Ideographic Description Sequences.
Join_Control	B	N	Format control characters which have specific functions for control of cursive joining and ligation.
Logical_Order_Exception	B	N	A small number of spacing vowel letters occurring in certain Southeast Asian scripts such as Thai and Lao, which use a visual order display model. These letters are stored in text ahead of syllable-initial consonants, and require special handling for processes such as searching and sorting.
Modifier_Combining_Mark	B	N	Arabic combining marks potentially reordered by the AMTRA algorithm specified in UAX #53.
Noncharacter_Code_Point	B	N	Code points permanently reserved for internal use.
Other_Alphabetic	B	C	Used in deriving the Alphabetic property.
Other_Default_Ignorable_Code_Point	B	C	Used in deriving the Default_Ignorable_Code_Point property.
Other_Grapheme_Extend	B	C	Used in deriving the Grapheme_Extend property.
Other_ID_Continue	B	C	Used to maintain backward compatibility of ID_Continue.
Other_ID_Start	B	C	Used to maintain backward compatibility of ID_Start.
Other_Lowercase	B	C	Used in deriving the Lowercase property.
Other_Math	B	C	Used in deriving the Math property.
Other_Uppercase	B	C	Used in deriving the Uppercase property.

Pattern_Syntax	B	N	Used for pattern syntax as described in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax" [UAX31].
Pattern_White_Space	B	N	
Prepended_Concatenation_Mark	B	I	A small class of visible format controls, which precede and then span a sequence of other characters, usually digits. These have also been known as "subtending marks", because most of them take a form which visually extends underneath the sequence of following digits.
Quotation_Mark	B	I	Punctuation characters that function as quotation marks.
Radical	B	N	Used in the definition of Ideographic Description Sequences.
Regional_Indicator	B	N	Property of the regional indicator characters, U+1F1E6..U+1F1FF. This property is referenced in various segmentation algorithms, to assist in correct breaking around emoji flag sequences.
Sentence_Terminal	B	I	Punctuation characters that generally mark the end of sentences. Used in Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29].
Soft_Dotted	B	N	Characters with a "soft dot", like *i* or *j*. An accent placed on these characters causes the dot to disappear. An explicit *dot above* can be added where required, such as in Lithuanian. See *Section 7.1, Latin* in [Unicode].
Terminal_Punctuation	B	I	Punctuation characters that generally mark the end of textual units.
Unified_Ideograph	B	N	A property which specifies the exact set of Unified CJK Ideographs in the standard. This set excludes CJK Compatibility Ideographs (which have canonical decompositions to Unified CJK Ideographs), as well as characters from the CJK Symbols and Punctuation block. The class of Unified_Ideograph=Y characters is a proper subset of the class of Ideographic=Y characters.
Variation_Selector	B	N	Indicates characters that are Variation Selectors. For details on the behavior of these characters, see *Section 23.4, Variation Selectors* in [Unicode], and Unicode Technical Standard #37, "Unicode Ideographic Variation Database" [UTS37].
White_Space	B	N	Spaces, separator characters and other control characters which should be treated by programming languages as "white space" for the purpose of parsing elements. See also Line_Break, Grapheme_Cluster_Break, Sentence_Break, and Word_Break, which classify space characters and related controls somewhat differently for particular text segmentation contexts.
UnicodeData.txt			
Name	M	N	(1) When a string value not enclosed in <angle brackets> occurs in this field, it specifies the character's Name property value, which matches exactly the name published in the code charts. The Name property value for most ideographic characters and for Hangul syllables is derived instead by various rules. See *Section 4.8, Name* in [Unicode] for a full specification of those rules. Strings enclosed in <angle brackets> in this field either provide label information used in the name derivation rules, or—in the case of characters which have a null string as their Name property value, such as control characters—provide other information about their code point type.
General_Category	E	N	(2) This is a useful breakdown into various character types which can be used as a default categorization in implementations. For the property values, see General Category Values.
Canonical_Combining_Class	N	N	(3) The classes used for the Canonical Ordering Algorithm in the Unicode Standard. This property could be considered either an enumerated property or a numeric property: the principal use of the property is in terms of the numeric values. For the property value names associated with different numeric values, see DerivedCombiningClass.txt and Canonical Combining Class Values.
Bidi_Class	E	N	(4) These are the categories required by the Unicode Bidirectional Algorithm. For the property values, see Bidirectional Class Values. For more information, see Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9]. The default property values depend on the code point, and are explained in DerivedBidiClass.txt
Decomposition_Type Decomposition_Mapping	E, S	N	(5) This field contains both values, with the type in angle brackets. The decomposition mappings exactly match the decomposition mappings published with the character names in the Unicode Standard. For more information, see Character Decomposition Mappings.
Numeric_Type Numeric_Value	E, N	N	(6) If the character has the property value Numeric_Type=Decimal, then the Numeric_Value of that digit is represented with an integer value (limited to the range 0..9) in fields 6, 7, and 8. Characters with the property value Numeric_Type=Decimal are restricted to digits which can be used in a decimal radix positional numeral system and which are encoded in the standard in a contiguous ascending range 0..9. See the discussion of *decimal digits* in *Chapter 4, Character Properties* in [Unicode].
	E, N	N	(7) If the character has the property value Numeric_Type=Digit, then the Numeric_Value of that digit is represented with an integer value (limited to the range 0..9) in fields 7 and 8, and field 6 is null. This covers digits that need special handling, such as the compatibility superscript digits. Starting with Unicode 6.3.0, no newly encoded numeric characters will be given Numeric_Type=Digit, nor will existing characters with Numeric_Type=Numeric be changed to Numeric_Type=Digit. The distinction between those two types is not considered useful.

	E, N	N	(8) If the character has the property value Numeric_Type=Numeric, then the Numeric_Value of that character is represented with a positive or negative integer or rational number in this field, and fields 6 and 7 are null. This includes fractions such as, for example, "1/5" for U+2155 VULGAR FRACTION ONE FIFTH. Some characters have these properties based on values from the Unihan data files. See Numeric_Type, Han.
Bidi_Mirrored	B	N	(9) If the character is a "mirrored" character in bidirectional text, this field has the value "Y"; otherwise "N". See *Section 4.7, Bidi Mirrored* of [Unicode]. *Do not confuse this with the Bidi_Mirroring_Glyph property.*
Unicode_1_Name (Obsolete as of 6.2.0)	M	I	(10) Old name as published in Unicode 1.0 or ISO 6429 names for control functions. This field is empty unless it is significantly different from the current name for the character. No longer used in code chart production. See Name_Alias.
ISO_Comment (Obsolete as of 5.2.0; Deprecated and Stabilized as of 6.0.0)	M	I	(11) ISO 10646 comment field. It was used for notes that appeared in parentheses in the 10646 names list, or contained an asterisk to mark an Annex P note. As of Unicode 5.2.0, this field no longer contains any non-null values.
Simple_Uppercase_Mapping	S	N	(12) Simple uppercase mapping (single character result). If a character is part of an alphabet with case distinctions, and has a simple uppercase equivalent, then the uppercase equivalent is in this field. The simple mappings have a single character result, where the full mappings may have multi-character results. For more information, see Case and Case Mapping.
Simple_Lowercase_Mapping	S	N	(13) Simple lowercase mapping (single character result).
Simple_Titlecase_Mapping	S	N	(14) Simple titlecase mapping (single character result). **Note:** If this field is null, then the Simple_Titlecase_Mapping is the same as the Simple_Uppercase_Mapping for this character.
emoji-data.txt			
Emoji	B	N	= **Yes** for characters that are emoji.
Emoji_Presentation	B	N	= **Yes** for characters that have emoji presentation by default.
Emoji_Modifier	B	N	= **Yes** for characters that are emoji modifiers. Currently this includes only the skin tone modifier characters.
Emoji_Modifier_Base	B	N	= **Yes** for characters that can serve as a base for emoji modifiers.
Emoji_Component	B	N	= **Yes** for characters used in emoji sequences that normally do not appear on emoji keyboards as separate choices, such as base characters for emoji keycaps. Also included are Regional_Indicator characters and U+FE0F VARIATION SELECTOR-16. **Note:** All characters in emoji sequences are either Emoji=Yes or Emoji_Component=Yes. However, implementations must not assume that all Emoji_Component=Yes characters are also Emoji=Yes. There are some non-emoji characters that are used in various emoji sequences, such as tag characters and ZWJ.
Extended_Pictographic	B	N	= **Yes** for pictographic symbols, as well as reserved ranges in blocks largely associated with emoji characters. This enables segmentation rules involving emoji to be specified stably, even in cases where an existing non-emoji pictographic symbol later comes to be treated as an emoji. **Note:** This property is used in the regex definitions for the Default Grapheme Cluster Boundary Specification and in rule GB11 in UAX #29, *Unicode Text Segmentation* [UAX29], in rule LB30b in UAX #14, *Unicode Line Breaking Algorithm* [UAX14], as well as for the definition ED-4 in UTS #51, *Unicode Emoji* [UTS51].

5.4 Derived Extracted Properties

A number of Unicode character properties have been separated out, reformatted, and listed in range format, one property per file. These files are located under the *extracted* directory of the UCD. The exact list of derived extracted files and the extracted properties they represent are given in *Table 10*.

The derived extracted files are provided primarily as a reformatting of data for properties specified in other data files. For *nondefault* values of properties, if there is any inadvertent mismatch between the primary data files specifying those properties and these lists of extracted properties, the primary data files are taken as definitive. However, for *default* values of properties, the extracted data files are definitive. This is particularly true for properties which have multiple default values; those properties are identified with an asterisk in the table. See Section 4.2.9, Default Values.

Table 10. Extracted Properties

File	Status	Property	Extracted from
DerivedBidiClass.txt	N	Bidi_Class*	UnicodeData.txt, field 4
DerivedBinaryProperties.txt	N	Bidi_Mirrored	UnicodeData.txt, field 9
DerivedCombiningClass.txt	N	Canonical_Combining_Class	UnicodeData.txt, field 3

DerivedDecompositionType.txt	N/I	Decomposition_Type	the <tag> in UnicodeData.txt, field 5
DerivedEastAsianWidth.txt	I	East_Asian_Width*	EastAsianWidth.txt, field 1
DerivedGeneralCategory.txt	N	General_Category	UnicodeData.txt, field 2
DerivedJoiningGroup.txt	N	Joining_Group	ArabicShaping.txt, field 3
DerivedJoiningType.txt	N	Joining_Type*	ArabicShaping.txt, field 2
DerivedLineBreak.txt	N	Line_Break*	LineBreak.txt, field 1
DerivedName.txt	N	Name	UnicodeData.txt, field 1
DerivedNumericType.txt	N	Numeric_Type	UnicodeData.txt, fields 6 through 8
DerivedNumericValues.txt	N	Numeric_Value	UnicodeData.txt, field 8

For the extraction of Decomposition_Type, characters with canonical decomposition mappings in field 5 of UnicodeData.txt have no tag. For those characters, the extracted value is Decomposition_Type=Canonical. For characters with compatibility decomposition mappings, there are explicit tags in field 5, and the value of Decomposition_Type is equivalent to those tags. The value Decomposition_Type=Canonical is normative. Other values for Decomposition_Type are informative.

The value of the Name property is extracted based on the actual string value of the data in field 1 of UnicodeData.txt, omitting any code points with the default null string value. Then for code points in the Hangul Syllables block, the Hangul Syllable Name Generation algorithm defined in *Section 3.12, Conjoining Jamo Behavior* of [Unicode] is applied, to create the explicit formal names of all Hangul syllables. Characters whose names are algorithmically defined based on suffixing the code point to a specific identifying string prefix, such as CJK UNIFIED IDEOGRAPH-4E00, are listed with a compact range convention in DerivedName.txt, using an asterisk "*" character as the placeholder for the code point. See *Section 4.8, Name* of [Unicode] for more information about how the Name property is derived.

Numeric_Value is extracted based on the actual numeric value of the data in field 8 of UnicodeData.txt or the first of the values of the kPrimaryNumeric, kAccountingNumeric, or kOtherNumeric tags, for characters listed in the Unihan data files.

Numeric_Type is extracted as follows. If fields 6, 7, and 8 in UnicodeData.txt are all non-empty, then Numeric_Type=Decimal. Otherwise, if fields 7 and 8 are both non-empty, then Numeric_Type=Digit. Otherwise, if field 8 is non-empty, then Numeric_Type=Numeric. For characters listed in the Unihan data files, Numeric_Type=Numeric for characters that have kPrimaryNumeric, kAccountingNumeric, or kOtherNumeric tags. The default value is Numeric_Type=None.

5.5 Contributory Properties

Contributory properties contain sets of exceptions used in the generation of other properties derived from them. The contributory properties specifically concerned with identifiers and casing contribute to the maintenance of stability guarantees for properties and/or to invariance relationships between related properties. Other contributory properties are simply defined as a convenience for property derivation.

Most contributory properties have names using the pattern "Other_XXX" and are used to derive the corresponding "XXX" property. For example, the Other_Alphabetic property is used in the derivation of the Alphabetic property.

Contributory properties are typically defined in PropList.txt and the corresponding derived property is then listed in DerivedCoreProperties.txt.

Jamo_Short_Name is an unusual contributory property, both in terms of its name and how it is used. It is defined in its own property file, Jamo.txt, and is used to derive the Name property value for Hangul syllable characters, according to the rules spelled out in *Section 3.12, Conjoining Jamo Behavior* in [Unicode].

Contributory is considered to be a distinct status for a Unicode character property. Contributory properties are neither *normative* nor *informative*. This distinct status is marked with the symbol "C" in the status column in the property table. For convenience of reference, all contributory properties are also listed in *Table 10a*, along with the properties whose derivation they contribute to.

Table 10a. Contributory Properties

File	Property	Used in Derivation of
Jamo.txt	Jamo_Short_Name	Name
PropList.txt	Other_Alphabetic	Alphabetic
	Other_Default_Ignorable_Code_Point	Default_Ignorable_Code_Point
	Other_Grapheme_Extend	Grapheme_Extend
	Other_ID_Start	ID_Start, XID_Start
	Other_ID_Continue	ID_Continue, XID_Continue
	Other_Lowercase	Lowercase
	Other_Math	Math
	Other_Uppercase	Uppercase

Contributory properties are incomplete by themselves and are not intended for independent use. For example, an API returning Unicode property values should implement the derived core properties such as Alphabetic or Default_Ignorable_Code_Point, rather than the corresponding contributory properties, Other_Alphabetic or Other_Default_Ignorable_Code_Point.

5.6 Case and Case Mapping

Case for bicameral scripts and case mapping of characters are complicated topics in the Unicode Standard—both because of their inherent algorithmic complexity and because of the number of characters and special edge cases involved.

This section provides a brief roadmap to discussions about these topics, and specifications and definitions in the standard, as well as explaining which case-related properties are defined in the UCD.

Section 3.13, Default Case Algorithms in [Unicode] provides formal definitions for a number of case-related concepts (*cased*, *case-ignorable*, ...), for case conversion (*toUppercase(X)*, ...), and for case detection (*isUppercase(X)*, ...). It also provides the formal definition of caseless matching for the standard, taking normalization into account.

Section 4.2, Case in [Unicode] introduces case and case mapping properties. *Table 4-3, Sources for Case Mapping Information* in [Unicode] describes the kind of case-related information that is available in various data files of the UCD. *Table 11* lists those data files again, giving the explicit list of case-related properties defined in each. The link on each property leads its description in *Table 9, Property Table*.

Table 11. UCD Files and Case Properties

File Name	Case Properties
UnicodeData.txt	Simple_Uppercase_Mapping, Simple_Lowercase_Mapping, Simple_Titlecase_Mapping
SpecialCasing.txt	Uppercase_Mapping, Lowercase_Mapping, Titlecase_Mapping
CaseFolding.txt	Simple_Case_Folding, Case_Folding
DerivedCoreProperties.txt	Uppercase, Lowercase, Cased, Case_Ignorable, Changes_When_Lowercased, Changes_When_Uppercased, Changes_When_Titlecased, Changes_When_Casefolded, Changes_When_Casemapped
DerivedNormalizationProps.txt	NFKC_Casefold, Changes_When_NFKC_Casefolded
PropList.txt	Soft_Dotted, Other_Uppercase, Other_Lowercase

For compatibility with existing parsers, UnicodeData.txt only contains case mappings for characters where they constitute one-to-one mappings; it also omits information about context-sensitive case mappings. Information about these special cases can be found in the separate data file, SpecialCasing.txt, expressed as separate properties.

Section 5.18, Case Mappings, in [Unicode] discusses various implementation issues for handling case, including language-specific case mapping, as for Greek and for Turkish. That section also describes case folding in particular detail.

The special casing conditions associated with case mapping for Greek, Turkish, and Lithuanian are specified in an additional field in SpecialCasing.txt. For example, the lowercase mapping for sigma in Greek varies according to its position in a word. The condition list does not constitute a formal character property in the UCD, because it is a statement about the context of occurrence of casing behavior for a character or characters, rather than a semantic attribute of those characters. Versions of the UCD from Version 3.2.0 to Version 5.0.0 *did* list property aliases for Special_Case_Condition (scc), but this was determined to be an error when the UCD was analyzed for representation in XML; consequently, the Special_Case_Condition property aliases were removed as of Version 5.1.0.

Caseless matching is of particular concern for a number of text processing algorithms, so is also discussed at some length in Unicode Standard Annex #31, "Unicode Identifier and Pattern Syntax" [UAX31] and in Unicode Technical Standard #10, "Unicode Collation Algorithm" [UTS10].

Further information about locale-specific casing conventions can be found in the Unicode Common Locale Data Repository [CLDR].

5.7 Property Value Lists

The following subsections give summaries of property values for certain Enumeration properties. Other property values are documented in other, topically-specific annexes; for example, the Line_Break property values are documented in Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14] and the various segmentation-related property values are documented in Unicode Standard Annex #29, "Unicode Text Segmentation" [UAX29].

5.7.1 General Category Values

The General_Category property of a code point provides for the most general classification of that code point. It is usually determined based on the primary characteristic of the assigned character for that code point. For example, is the character a letter, a mark, a number, punctuation, or a symbol, and if so, of what type? Other General_Category values define the classification of code points which are not assigned to regular graphic characters, including such statuses as private-use, control, surrogate code point, and reserved unassigned.

Many characters have multiple uses, and not all such cases can be captured entirely by the General_Category value. For example, the General_Category value of Latin, Greek, or Hebrew letters does not attempt to cover (or preclude) the numerical use of such letters as Roman numerals or in other numerary systems. Conversely, the General_Category of ASCII digits 0..9 as Nd (decimal digit) neither attempts to cover (or preclude) the occasional use of these digits as letters in various orthographies. The General_Category is simply the first-order, most usual categorization of a character.

For more information about the General_Category property, see *Chapter 4, Character Properties* in [Unicode].

The values in the General_Category field in UnicodeData.txt make use of the short, abbreviated property value aliases for General_Category. For convenience in reference, *Table 12* lists all the abbreviated and long value aliases for General_Category values, reproduced from PropertyValueAliases.txt, along with a brief description of each category.

Table 12. General_Category Values

Abbr	Long	Description
Lu	Uppercase_Letter	an uppercase letter
Ll	Lowercase_Letter	a lowercase letter
Lt	Titlecase_Letter	a digraph encoded as a single character, with first part uppercase

LC	Cased_Letter	Lu | Ll | Lt
Lm	Modifier_Letter	a modifier letter
Lo	Other_Letter	other letters, including syllables and ideographs
L	Letter	Lu | Ll | Lt | Lm | Lo
Mn	Nonspacing_Mark	a nonspacing combining mark (zero advance width)
Mc	Spacing_Mark	a spacing combining mark (positive advance width)
Me	Enclosing_Mark	an enclosing combining mark
M	Mark	Mn | Mc | Me
Nd	Decimal_Number	a decimal digit
Nl	Letter_Number	a letterlike numeric character
No	Other_Number	a numeric character of other type
N	Number	Nd | Nl | No
Pc	Connector_Punctuation	a connecting punctuation mark, like a tie
Pd	Dash_Punctuation	a dash or hyphen punctuation mark
Ps	Open_Punctuation	an opening punctuation mark (of a pair)
Pe	Close_Punctuation	a closing punctuation mark (of a pair)
Pi	Initial_Punctuation	an initial quotation mark
Pf	Final_Punctuation	a final quotation mark
Po	Other_Punctuation	a punctuation mark of other type
P	Punctuation	Pc | Pd | Ps | Pe | Pi | Pf | Po
Sm	Math_Symbol	a symbol of mathematical use
Sc	Currency_Symbol	a currency sign
Sk	Modifier_Symbol	a non-letterlike modifier symbol
So	Other_Symbol	a symbol of other type
S	Symbol	Sm | Sc | Sk | So
Zs	Space_Separator	a space character (of various non-zero widths)
Zl	Line_Separator	U+2028 LINE SEPARATOR only
Zp	Paragraph_Separator	U+2029 PARAGRAPH SEPARATOR only
Z	Separator	Zs | Zl | Zp
Cc	Control	a C0 or C1 control code
Cf	Format	a format control character
Cs	Surrogate	a surrogate code point
Co	Private_Use	a private-use character
Cn	Unassigned	a reserved unassigned code point or a noncharacter
C	Other	Cc | Cf | Cs | Co | Cn

Note that the value gc=Cn does not actually occur in UnicodeData.txt, because that data file does not list unassigned code points.

The distinctions between some General_Category values are somewhat arbitrary for edge cases, particularly those involving symbols and punctuation. For example, a number of multiple-function ASCII characters, including "@", "#", "%", and "&", have long been classified as Other_Punctuation (gc=Po), although they are not among the characters used as punctuation marks in traditional Western typography. Other characters may also be ambiguous between functioning to organize and delimit textual units (punctuation-like) or to represent concepts (symbol-like). Likewise, it may not always be clear whether some symbols are primarily used for mathematics or whether they are general symbols with occasional or even common use in mathematics. For example, many arrow symbols are classed as Other_Symbol, although they are widely used in mathematics. The General_Category values constitute a rough partitioning of characters to make distinctions for algorithmic processing, but do not provide a definitive classification for such overlapping or ambiguous usage of characters.

Characters with the quotation-related General_Category values Pi or Pf may behave like opening punctuation (gc=Ps) or closing punctuation (gc=Pe), depending on usage and quotation conventions.

General_Category values in the table highlighted in light blue (LC, L, M, N, P, S, Z, C) stand for groupings of related General_Category values. The classes they represent can be derived by unions of the relevant simple values, as shown in the table. The abbreviated and long value aliases for these classes are provided as a convenience for implementations, such as regex, which may wish to match more generic categories, such as "letter" or "number", rather than the detailed subtypes for General_Category. These aliases for groupings of General_Category values do not occur in UnicodeData.txt, which instead always specifies the enumerated subtype for the General_Category of a character.

The symbol "L&" is a label used to stand for any combination of uppercase, lowercase or titlecase letters (Lu, Ll, or Lt), in the first part of comments in the data files of the UCD. It is equivalent to gc=LC, but is only a label in comments, and is not expected to be used as an identifier for regular expression matching.

The Unicode Standard does not assign nondefault property values to control characters (gc=Cc), except for certain well-defined exceptions involving the Unicode Bidirectional Algorithm, the Unicode Line Breaking Algorithm, and Unicode Text Segmentation. Also, implementations will usually assign behavior to certain line breaking control characters—most notably U+000D and U+000A (CR and LF)—according to platform conventions. See *Section 5.8, Newline Guidelines* in [Unicode] for more information.

5.7.2 Bidirectional Class Values

The values in the Bidi_Class field in UnicodeData.txt make use of the short, abbreviated property value aliases for Bidi_Class. For convenience in reference, *Table 13* lists all the abbreviated and long value aliases for Bidi_Class values, reproduced from PropertyValueAliases.txt, along with a brief description of each category.

Table 13. Bidi_Class Values

Abbr	Long	Description
Strong Types		
L	Left_To_Right	any strong left-to-right character
R	Right_To_Left	any strong right-to-left (non-Arabic-type) character
AL	Arabic_Letter	any strong right-to-left (Arabic-type) character
Weak Types		
EN	European_Number	any ASCII digit or Eastern Arabic-Indic digit
ES	European_Separator	plus and minus signs
ET	European_Terminator	a terminator in a numeric format context, includes currency signs
AN	Arabic_Number	any Arabic-Indic digit
CS	Common_Separator	commas, colons, and slashes
NSM	Nonspacing_Mark	any nonspacing mark
BN	Boundary_Neutral	most format characters, control codes, or noncharacters
Neutral Types		
B	Paragraph_Separator	various newline characters
S	Segment_Separator	various segment-related control codes
WS	White_Space	spaces
ON	Other_Neutral	most other symbols and punctuation marks
Explicit Formatting Types		
LRE	Left_To_Right_Embedding	U+202A: the LR embedding control
LRO	Left_To_Right_Override	U+202D: the LR override control
RLE	Right_To_Left_Embedding	U+202B: the RL embedding control
RLO	Right_To_Left_Override	U+202E: the RL override control
PDF	Pop_Directional_Format	U+202C: terminates an embedding or override control
LRI	Left_To_Right_Isolate	U+2066: the LR isolate control
RLI	Right_To_Left_Isolate	U+2067: the RL isolate control
FSI	First_Strong_Isolate	U+2068: the first strong isolate control
PDI	Pop_Directional_Isolate	U+2069: terminates an isolate control

Please refer to Unicode Standard Annex #9, "Unicode Bidirectional Algorithm" [UAX9] for an an explanation of the significance of these values when formatting bidirectional text.

The four enumerated values for the isolate controls were added in Unicode 6.3. That means there is a discontinuity in the enumeration for Bidi_Class between Unicode 6.2 and Unicode 6.3 (and later versions) which parsers of UnicodeData.txt and DerivedBidiClass.txt must take into account.

5.7.3 Character Decomposition Mapping

The value of the Decomposition_Mapping property for a character is provided in field 5 of UnicodeData.txt. This is a string-valued property, consisting of a sequence of one or more Unicode code points. The default value of the Decomposition_Mapping property is the code point of the character itself. The use of the default value for a character is indicated by leaving field 5 empty in UnicodeData.txt. Informally, the value of the Decomposition_Mapping property for a character is known simply as its *decomposition mapping*. When a character's decomposition mapping is other than the default value, the decomposition mapping is printed out explicitly in the names list for the Unicode code charts.

The prefixed tags supplied with a subset of the decomposition mappings generally indicate formatting information. Where no such tag is given, the mapping is canonical. Conversely, the presence of a formatting tag also indicates that the mapping is a compatibility mapping and not a canonical mapping. In the absence of other formatting information in a compatibility mapping, the tag is used to distinguish it from canonical mappings.

In some instances a canonical mapping or a compatibility mapping may consist of a single character. For a canonical mapping, this indicates that the character is a canonical equivalent of another single character. For a compatibility mapping, this indicates that the character is a

compatibility equivalent of another single character.

A canonical mapping may also consist of a pair of characters, but is never longer than two characters. When a canonical mapping consists of a pair of characters, the first character may itself be a character with a decomposition mapping, but the second character never has a decomposition mapping.

Compatibility mappings can be much longer than canonical mappings. For historical reasons, the longest compatibility mapping is 18 characters long. Compatibility mappings are guaranteed to be no longer than 18 characters, although most consist of just a few characters.

The compatibility formatting tags used in the UCD are listed in *Table 14*.

Table 14. Compatibility Formatting Tags

Tag	Description
<font>	Font variant (for example, a blackletter form)
<noBreak>	No-break version of a space or hyphen
<initial>	Initial presentation form (Arabic)
<medial>	Medial presentation form (Arabic)
<final>	Final presentation form (Arabic)
<isolated>	Isolated presentation form (Arabic)
<circle>	Encircled form
<super>	Superscript form
<sub>	Subscript form
<vertical>	Vertical layout presentation form
<wide>	Wide (or zenkaku) compatibility character
<narrow>	Narrow (or hankaku) compatibility character
<small>	Small variant form (CNS compatibility)
<square>	CJK squared font variant
<fraction>	Vulgar fraction form
<compat>	Otherwise unspecified compatibility character

Note: There is a difference between decomposition and the Decomposition_Mapping property. The Decomposition_Mapping property is a string-valued property whose values (mappings) are defined in UnicodeData.txt, while the decomposition (also termed "full decomposition") is defined in *Section 3.7, Decomposition* in [Unicode] to use those mappings *recursively.*

- The canonical decomposition is formed by recursively applying the canonical mappings, then applying the Canonical Ordering Algorithm.
- The compatibility decomposition is formed by recursively applying the canonical **and** compatibility mappings, then applying the Canonical Ordering Algorithm.

Starting from Unicode 2.1.9, the decomposition mappings in UnicodeData.txt can be used to derive the full decomposition of any single character in canonical order, without the need to separately apply the Canonical Ordering Algorithm. However, canonical ordering of combining character sequences ***must*** still be applied in decomposition when normalizing source text which contains any combining marks.

The normalization of Hangul conjoining jamos and of Hangul syllables depends on algorithmic mapping, as specified in *Section 3.12, Conjoining Jamo Behavior* in [Unicode]. That algorithm specifies the full decomposition of all precomposed Hangul syllables, but effectively it is equivalent to the recursive application of pairwise decomposition mappings, as for all other Unicode characters. Formally, the Decomposition_Mapping property value for a Hangul syllable is the pairwise decomposition and not the full decomposition.

Each character with the Hangul_Syllable_Type value LVT will have a Decomposition_Mapping consisting of a character with an LV value and a character with a T value. Thus for U+CE31 the Decomposition_Mapping is <U+CE20, U+11B8>, rather than <U+110E, U+1173, U+11B8>.

The Unihan property kCompatibilityVariant consists of a listing of the canonical Decomposition_Mapping property values just for CJK compatibility ideographs. Because its values are derived from UnicodeData.txt, it is formally considered to be a derived property. The exact statement of the derivation for kCompatibilityVariant is listed in Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38].

5.7.4 Canonical Combining Class Values

The values in the Canonical_Combining_Class field in UnicodeData.txt are numerical values used in the Canonical Ordering Algorithm. Some of those numerical values also have explicit symbolic labels as property value aliases, to make their intended application more understandable. For convenience in reference, *Table 15* lists the long symbolic aliases for Canonical_Combining_Class values, reproduced from PropertyValueAliases.txt, along with a brief description of each category. The listing for fixed position classes, with long symbolic aliases of the form "Ccc10", and so forth, is abbreviated, as when those labels occur they are predictable in form, based on the numeric values.

Table 15. Canonical_Combining_Class Values

Value	Long	Description
0	Not_Reordered	Spacing and enclosing marks; also many vowel and consonant signs, even if nonspacing
1	Overlay	Marks which overlay a base letter or symbol
6	Han_Reading	Diacritic reading marks for CJK unified ideographs

7	Nukta	Diacritic nukta marks in Brahmi-derived scripts
8	Kana_Voicing	Hiragana/Katakana voicing marks
9	Virama	Viramas
10	Ccc10	Start of fixed position classes
...	...	
199		End of fixed position classes
200	Attached_Below_Left	Marks attached at the bottom left
202	Attached_Below	Marks attached directly below
204		Marks attached at the bottom right
208		Marks attached to the left
210		Marks attached to the right
212		Marks attached at the top left
214	Attached_Above	Marks attached directly above
216	Attached_Above_Right	Marks attached at the top right
218	Below_Left	Distinct marks at the bottom left
220	Below	Distinct marks directly below
222	Below_Right	Distinct marks at the bottom right
224	Left	Distinct marks to the left
226	Right	Distinct marks to the right
228	Above_Left	Distinct marks at the top left
230	Above	Distinct marks directly above
232	Above_Right	Distinct marks at the top right
233	Double_Below	Distinct marks subtending two bases
234	Double_Above	Distinct marks extending above two bases
240	Iota_Subscript	Greek iota subscript only

Some of the Canonical_Combining_Class values in the table are not currently used for any characters but are specified here for completeness. Some values do not have long symbolic aliases and are not listed in PropertyValueAliases.txt. Do not assume that absence of a long symbolic alias implies non-use of a particular Canonical_Combining_Class. See DerivedCombiningClass.txt for a complete listing of the use of Canonical_Combining_Class values for any particular version of the UCD.

For use in regular expression matching, fixed position classes (ccc=10 through ccc=199) which actually occur in the Unicode Character Database for any version are given predictable aliases of the form "Ccc10", "Ccc11", and so forth. The complete list of such aliases which are actually defined can be found in PropertyValueAliases.txt.

The character property invariants regarding Canonical_Combining_Class guarantee that values, once assigned, will never change, and that all values used will be in the range 0..254. See Invariants in Implementations.

Combining marks with ccc=224 (Left) follow their base character in storage, as for all combining marks, but are rendered visually on the left side of them. For all past versions of the UCD and continuing with this version of the UCD, only two tone marks used in certain notations for Hangul syllables have ccc=224. Those marks are actually rendered visually on the left side of the preceding *grapheme cluster*, in the case of Hangul syllables resulting from sequences of conjoining jamos.

Those few instances of combining marks with ccc=Left should be distinguished from the far more numerous examples of left-side vowel signs and vowel letters in Brahmi-derived scripts. The Canonical_Combining_Class value is zero (Not_Reordered) for both ordinary, left-side (reordrant) vowel signs such as U+093F DEVANAGARI VOWEL SIGN I and for Thai-style left-side (Logical_Order_Exception=Yes) vowel letters such as U+0E40 THAI CHARACTER SARA E. The "Not_Reordered" of ccc=Not_Reordered refers to the behavior of the character in terms of the Canonical Ordering Algorithm as part of the definition of Unicode Normalization; it does *not* refer to any issues of visual reordering of glyphs involved in display and rendering. See "Canonical Ordering Algorithm" in *Section 3.11, Normalization Forms* in [Unicode].

5.7.5 Decompositions and Normalization

Decomposition is specified in *Chapter 3, Conformance* of [Unicode]. That chapter also specifies the interaction between decomposition and normalization.

A number of derived properties related to Unicode normalization are called the "Quick_Check" properties. These are defined to enable various optimizations for implementations of normalization, as explained in *Section 9, Detecting Normalization Forms*, in Unicode Standard Annex #15, "Unicode Normalization Forms" [UAX15]. The values for the four Quick_Check properties for all code points are listed in DerivedNormalizationProps.txt. The interpretations of the possible property values are summarized in *Table 16*.

Table 16. Quick_Check Property Values

Property	Value	Description
NFC_QC, NFKC_QC, NFD_QC, NFKD_QC	No	Characters that cannot ever occur in the respective normalization form.

NFC_QC, NFKC_QC	Maybe	Characters that may occur in the respective normalization, depending on the context.
NFC_QC, NFKC_QC, NFD_QC, NFKD_QC	Yes	All other characters. This is the default value for Quick_Check properties.

The Quick_Check property values are recommended for exposure in a public library API which supports Unicode character properties, because they can be used to optimize code that needs to normalize Unicode strings. They enable fast checking of whether some input strings are already in the desired normalization form. This may make it possible to bypass the more time-consuming call to run the complete Unicode Normalization Algorithm on the input string.

In contrast, some normalization-related Unicode character properties are *not* recommended for exposure in a public library API. Notably, these include Decomposition_Mapping, Composition_Exclusion, and the derived Full_Composition_Exclusion. These properties are only used internally in a conformant implementation of the Unicode Normalization Algorithm. Exposing them in a public API can lead to confusion by users of the API. In particular, Decomposition_Mapping is very easy to misinterpret as designating the *decomposition* of a character, also known as the character's *full decomposition*. See Definitions D62 and D64 in *Section 3.7, Decomposition* in [Unicode].

5.7.6 Properties Whose Values Are Sets of Values

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. For example, the provisional kVietnamese property, which lists Vietnamese pronunciations for unified CJK ideographs, has values which consist of a set of zero or more pronunciation strings. Thus, the Unihan Database contains an entry:

```
U+6258  kVietnamese thác thách thốc thước thướt
```

This line is to be interpreted as associating a set of five string values, {"thác", "thách", "thốc", "thước", "thướt"} with the kVietnamese property for U+6258.

Similarly, the Script_Extensions property has values which consist of a set of one or more Script property values. Thus the property file ScriptExtensions.txt in the UCD contains an entry:

```
0640          ; Adlm Arab Mand Mani Phlp Rohg Sogd Syrc # Lm       ARABIC TATWEEL
```

This line is to be interpreted as associating a set of eight enumerated Script property values, {Adlm, Arab, Mand, Mani, Phlp, Rohg, Sogd, Syrc}, with the Script_Extensions property for U+0640.

In the case of Script_Extensions, in particular, the set of sets which constitute meaningful values of the property is relatively small, and could be explicitly evaluated for any particular Unicode version. For example:

```
{{Adlm, Arab, Mand, Mani, Phlp, Rohg, Sogd, Syrc}, {Arab, Copt}, {Arab, Rohg}, {Arab, Syrc}, {Arab, Thaa}, {Arab, Syrc, Thaa}, {Armn, Geor}, ...}
```

However, an enumeration of this set of set values is unlikely to be of much implementation value, and would be likely to change significantly between versions of the standard. In other cases, such as for properties defining pronunciation readings for unified CJK ideographs, these sets of sets are completely open-ended, and there is no point to attempting to provide explicit enumerations of such sets in the UCD.

The order of the element values in such sets may or may not be significant. For example, the order among the element values for kCantonese and for Script_Extensions is not significant. By way of contrast, when the kMandarin property shows two values for a code point, the first value is used to indicate a preferred pronunciation for zh-Hans (CN) and the second a preferred pronunciation for zh-Hant (TW).

For data file format considerations regarding properties which take sets of values, see Section 4.2.8 Multiple Values for Properties. For considerations regarding validation of such properties, see Section 5.11.5 Validation of Multivalued Properties. See also Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18] for a discussion of how to handle such properties when processing regular expressions.

5.8 Property and Property Value Aliases

Both Unicode character properties themselves and their values are given symbolic aliases. The formal lists of aliases are provided so that well-defined symbolic values are available for XML formats of the UCD data, for regular expression property tests, and for other programmatic textual descriptions of Unicode data. The aliases for properties are defined in PropertyAliases.txt. The aliases for property values are defined in PropertyValueAliases.txt.

Table 17. Alias Files in the UCD

File Name	Status	Description
PropertyAliases.txt	N	Names and abbreviations for properties
PropertyValueAliases.txt	N	Names and abbreviations for property values

Aliases are defined as ASCII-compatible identifiers, using only uppercase or lowercase A-Z, digits, and underscore "_". Case is not significant when comparing aliases, but the preferred form used in the data files for longer aliases is to titlecase them.

Aliases may be translated in appropriate environments, and additional aliases may be useful in certain contexts. There is no requirement that only the aliases defined in the alias files of the UCD be used when referring to Unicode character properties or their values; however, their use is recommended for interoperability in data formats or in programmatic contexts.

Aliases may be provided for provisional properties. There are stability guarantees for property aliases and property value aliases, but no stability guarantees for provisional properties or other provisional data files; consequently, there can also be no stability guarantee for property aliases or property value aliases associated with provisional properties.

5.8.1 Property Aliases

In PropertyAliases.txt, the first field typically specifies an abbreviated symbolic name for the property, and the second field specifies the long symbolic name for the property. These are the preferred aliases. Additional aliases for a few properties are specified in the third or subsequent

fields.

Aliases for normative and informative properties defined in the Unihan data files are included in PropertyAliases.txt, beginning with Version 5.2.

The long symbolic name alias is self-descriptive, and is treated as the official name of a Unicode character property. For clarity it is used whenever possible when referring to that property in this annex and elsewhere in the Unicode Standard. For example: "The Line_Break property is discussed in Unicode Standard Annex #14, "Unicode Line Breaking Algorithm" [UAX14]."

The abbreviated symbolic name alias is usually short and less mnemonic, but is useful for expressions such as "lb=BA" in data or in other contexts where the meaning is clear. Note that although the UCD documentation refers to this first symbolic name alias as "abbreviated", there is no requirement that the first field be an actual abbreviation or even that it be shorter than the "long" symbolic name alias. If the long symbolic name alias is already a short identifier, in many cases the "abbreviated" symbolic name alias is identical to the value in the second field. There is also one principled class where the "abbreviated" field is actually longer than the "long" field—the property aliases for the Unihan tags. In that case, the second field deliberately matches the Unihan tags exactly, so that it can serve its function as being the official property value identifier. Then, because there was no systematic way to abbreviate Unihan tags, while still retaining any reasonable comprehensibility for them, the first field in PropertyAliases.txt was created by systematically prefixing "cj" to each Unihan tag, resulting in labels with the mnemonic "cjk" prefix. Thus it is not a mistake that in such cases the first field contains a longer string than the second field. Implementations should not build in assumptions about the relative length of these symbolic name aliases.

The property aliases specified in PropertyAliases.txt constitute a unique namespace. When using these symbolic values, no alias for one property will match an alias for another property.

5.8.2 Property Value Aliases

In PropertyValueAliases.txt, the first field contains the abbreviated alias for a Unicode property, the second field specifies an abbreviated symbolic name for a value of that property, and the third field specifies the long symbolic name for that value of that property. These are the preferred aliases. Additional aliases for some property values may be specified in the fourth or subsequent fields. For example, for binary properties, the abbreviated alias for the True value is "Y", and the long alias is "Yes", but each entry also specifies "T" and "True" as additional aliases for that value, as shown in *Table 18*.

Table 18. Binary Property Value Aliases

Long	Abbreviated	Other Aliases
Yes	Y	True, T
No	N	False, F

Not every property value has an associated alias. Property value aliases are typically supplied for catalog and enumeration properties, which have well-defined, enumerated values. It does not make sense to specify property value aliases, for example, for the Numeric_Value property, whose value could be any number, or for a string-valued property such as Simple_Lowercase_Mapping, whose values are mappings from one code point to another.

The Canonical_Combining_Class property requires special handling in PropertyValueAliases.txt. The values of this property are numeric, but they comprise a closed, enumerated set of values. The more important of those values are given symbolic name aliases. In PropertyValueAliases.txt, the second field provides the numeric value, while the third field contains the abbreviated symbolic name alias and the fourth field contains the long symbolic name alias for that numeric value. For example:

```
ccc; 230; A    ; Above
ccc; 232; AR   ; Above_Right
```

Taken by themselves, property value aliases do not constitute a unique namespace. The abbreviated aliases, in particular, are often re-used as aliases for values for different properties. All of the binary property value aliases, for example, make use of the same "Y", "Yes", "T", "True" symbols. Property value aliases may also overlap the symbols used for property aliases. For example, "Sc" is the abbreviated alias for the "Currency_Symbol" value of the General_Category property, but it is also the abbreviated alias for the Script property. However, the aliases for values for any single property are always unique within the context of that property. That means that expressions that combine a property alias and a property value alias, such as "lb=BA" or "gc=Sc" *always* refer unambiguously just to one value of one given property, and will not match any other value of any other property.

Prior to Version 6.1.0, the property value alias entries for three properties, Age, Block, and Joining_Group, made use of a special metavalue "n/a" in the field for the abbreviated alias. This should be understood as meaning that no abbreviated alias was defined for that value for that property, rather than as an alias per se. Starting with Version 6.1.0, all property values for those three properties have abbreviated aliases, so there is no current use of the "n/a" metavalue.

In a few cases, because of longstanding legacy practice in referring to values of a property by short identifiers, the abbreviated alias and the long alias are the same. This can be seen, for example, in some property value aliases for the Line_Break property and the Grapheme_Cluster_Break property.

The property Script_Extensions consists of enumerated sets of Script property values. The set of those sets is potentially open-ended, and no property value aliases are defined for them.

5.9 Matching Rules

When matching Unicode character property names and values, it is strongly recommended that all Property and Property Value Aliases be recognized. For best results in matching, rather than using exact binary comparisons, the following loose matching rules should be observed.

5.9.1 Matching Numeric Property Values

For all numeric properties, and for properties such as Unicode_Radical_Stroke which are constructed from combinations of numeric values, use loose matching rule UAX44-LM1 when comparing property values.

UAX44-LM1. Apply numeric equivalences.

- "01.00" is equivalent to "1".
- "1.666667" in the UCD is a repeating fraction, and equivalent to "10/6" or "5/3".

5.9.2 Matching Character Names

Unicode character names constitute a special case. Formally, they are values of the Name property. While each Unicode character name for an assigned character is guaranteed to be unique, names are assigned in such a way that the presence or absence of spaces cannot be used to distinguish them. Furthermore, implementations sometimes create identifiers from Unicode character names by inserting underscores for spaces. For best results in comparing Unicode character names, use loose matching rule UAX44-LM2.

UAX44-LM2. Ignore case, whitespace, underscore ('_'), and all medial hyphens except the hyphen in U+1180 HANGUL JUNGSEONG O-E.

- "zero-width space" is equivalent to "ZERO WIDTH SPACE" or "zerowidthspace"
- "character -a" is *not* equivalent to "character a"

In this rule "medial hyphen" is to be construed as a hyphen occurring immediately between two alphanumeric characters [A..Z, 0..9] in the normative Unicode character name, as published in the Unicode names list in the UCD, and not to any hyphen that may transiently occur medially as a result of removing whitespace before removing hyphens in a particular implementation of matching. (See *Section 4.8, Name* in [Unicode] for the normative specification of the Unicode Name property and of name uniqueness.)

Thus the hyphens in the following examples of character names are medial, and should be ignored in loose matching:

- U+10089 LINEAR B IDEOGRAM B107M HE-GOAT
- U+2F800 CJK COMPATIBILITY IDEOGRAPH-2F800
- U+1FB23 BLOCK SEXTANT-136
- U+10749 LINEAR A SIGN A709-2 L2
- U+1F090 DOMINO TILE VERTICAL-06-03

In contrast, the hyphens in the following examples of character names are *not* medial, and should not be ignored in loose matching.

- U+0F39 TIBETAN MARK TSA -PHRU
- U+11C88 MARCHEN LETTER -A

An implementation of this loose matching rule can obtain the correct results when comparing two strings by doing the following three operations, in order:

1. remove all medial hyphens (except the medial hyphen in the name for U+1180)
2. remove all whitespace and underscore characters
3. apply toLowercase() to both strings

After applying these three operations, if the two strings compare binary equal, then they are considered to match.

This is a logical statement of how the rule works. If programmed carefully, an implementation of the matching rule can transform the strings in a single pass. It is also possible to compare two name strings for loose matching while transforming each string incrementally.

Loose matching rule UAX44-LM2 is also appropriate for matching character name aliases, the names of named character sequences, and code point labels, which all share the unique namespace (and matching behavior) of Unicode character names. See *Section 4.8, Name* in [Unicode]

Examples of medial hyphens in character name aliases include:

- U+008E SINGLE-SHIFT-2
- U+11EC HANGUL JONGSEONG YESIEUNG-KIYEOK

Examples of *non*-medial hyphens in character name aliases include:

- U+0FD0 TIBETAN MARK BKA- SHOG GI MGO RGYAN

Examples of medial hyphens in named character sequences include:

- MODIFIER LETTER EXTRA-HIGH EXTRA-LOW CONTOUR TONE BAR;02E5 02E9

Implementations of name matching should use extreme care when matching non-standard, alternative names for particular characters. The Name Uniqueness Policy in the Unicode Consortium Stability Policies [Stability] guarantees that the Unicode Standard will never add a character whose name would match an existing encoded character, according to matching rule UAX44-LM2. However, any *other* name for a character might be used in the future.

The following is a concrete example of the kind of trouble that can occur. Prior to Unicode 6.0 some implementations of regex allowed matching of the name "BELL" for the control code U+0007. When Unicode 6.0 added a *different* encoded character, U+1F514 BELL for emoji symbols, those regex implementations broke.

As of Version 6.1 of the Unicode Standard, the most commonly occurring alternative names for control codes, as well as many commonly used abbreviations for Unicode format characters, have been added as character name aliases. This automatically excludes all such alternative names and abbreviations from the potential pool for future Unicode character names, because name uniqueness is defined over the namespace which includes both character names and character name aliases. That exclusion should reduce the potential for surprises similar to the "BELL" case, where implementers assume that a name for a control code is already well-defined.

5.9.3 Matching Symbolic Values

Property aliases and property value aliases are symbolic values. When comparing them, use loose matching rule UAX44-LM3.

UAX44-LM3. Ignore case, whitespace, underscore ('_'), hyphens, and any initial prefix string "is".

- "linebreak" is equivalent to "Line_Break" or "Line-break"
- "lb=BA" is equivalent to "lb=ba" or "LB=BA"
- "Script=Greek" is equivalent to "Script=isGreek" or "Script=Is_Greek"

Loose matching is generally appropriate for the property values of Catalog, Enumeration, and Binary properties, which have symbolic aliases defined for their values. Loose matching should not be done for the property values of String-valued properties, which do not have symbolic aliases defined for their values; exact matching for String-valued property values is important, as case distinctions or other distinctions in those values may be significant.

For loose matching of symbolic values, an initial prefix string "is" is ignored. The reason for this is that APIs returning property values are often named using the convention of prefixing "is" (or "Is" or "Is_", and so forth) to a property value. Ignoring any initial "is" on a symbolic value during loose matching is likely to produce the best results in application areas such as regex. Removal of an initial "is" string for a loose matching comparison only needs to be done once for a symbolic value, and need not be tested recursively. There are no property aliases or property value aliases of the form "isisisisistooconvoluted" defined just to test implementation edge cases.

Existing and future property aliases and property value aliases are guaranteed to be unique within their relevant namespaces, even if an initial prefix string "is" is ignored. The existing cases of note for aliases that do start with "is" are: dt=Iso (Decomposition_Type=Isolated) and lb=IS. The Decomposition_Type value alias does not cause any problem, because there is no contrasting value alias dt=o (Decomposition_Type=olated). For lb=IS, note that the "IS" is the *entire* property value alias, and is not a prefix. There is no null value for the Line_Break property for it to contrast with, but implementations of loose matching should be careful of this edge case, so that "lb=IS" is not misinterpreted as matching a null value.

Implementations sometimes use other syntactic constructs that interact with loose matching. For example, the property matching expression \p{L} may be defaulted to refer to the Unicode General_Category property: \p{General_Category=L}. For more information about the use of property values in regular expressions and other environments, see *Section 1.2, Properties*, in Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18].

5.10 Invariants

Property values in the UCD may be subject to correction in subsequent versions of the standard, as errors are found. Furthermore, any new version of the Unicode Standard may introduce new property values for a given property, except where the set of allowable values is fixed by the property type (such as for binary properties), or where the set of allowable values is subject to a provision of the Unicode Character Encoding Stability Policy [Stability]. Finally, a new version may also introduce new properties or new data files in the UCD.

Implementers of the UCD need to be aware of such changes when updating to new versions. However, some property values and some aspects of the file formats are considered invariant. This section documents such invariants.

5.10.1 Character Property Invariants

All formally guaranteed invariants for properties or property values are described in the Unicode Character Encoding Stability Policy [Stability]. That policy and the list of invariants it enumerates are maintained outside the context of the Unicode Standard per se. They are not part of the standard, but rather are constraints on what can and cannot change in the standard between versions, and on what decisions the Unicode Technical Committee can and cannot take regarding the standard.

In addition to the formally guaranteed invariants described in the Unicode Character Encoding Stability Policy, this section notes a few additional points regarding character property invariants in the UCD.

Some character properties are simply considered *immutable*: once assigned, they are never changed. For example, a character's name is immutable, because of its importance in exact identification of the character. The Canonical_Combining_Class and Decomposition_Mapping of a character are immutable, because of their importance to the stability of the Unicode Normalization Algorithm [UAX15].

The list of immutable character properties is shown in *Table 19*.

Table 19. Immutable Properties

Property Name	Abbr Name	Default Value	Assignable to New?
Age	Age	Unassigned	Yes
Name	na	null string	Yes
Name_Alias	Name_Alias	null string	Yes (see note)
Jamo_Short_Name	jsn	null string	No
Canonical_Combining_Class	ccc	0	Yes
Decomposition_Mapping	dm	<code point>	Yes
Pattern_Syntax	Pat_Syn	No	No
Pattern_White_Space	Pat_WS	No	No
Noncharacter_Code_Point	NChar	No	No

If a property has "Yes" in the "Assignable to New?" column in *Table 19*, that means that the property value is immutable once it is initially assigned to a newly encoded character. The value for a reserved code point takes the default value, as shown in the third column of the table, but *may change* from the default value once the character is encoded. On the other hand, if a property has "No" in the "Assignable to New?" column, that means that it is *absolutely* immutable: all code points, including reserved code points, have a specific property value assigned, and that value does not change if a new character is encoded at a particular reserved code point in a future version of the standard.

The Name_Alias property is unusual, in that there can be more than one formal name alias assigned to a given encoded character. The default value for Name_Alias is the null string, but once any Name_Alias is assigned to an encoded character, that value is immutable. If more than one formal name alias is assigned to the same encoded character, each of those values is immutable.

A set of binary character properties associated with identifiers have a different kind of immutability, which can be described as *locked to Yes*. This results from the way these properties are used in the specification of identifiers. Unicode identifiers have the characteristic of stability between versions, so that once a string is specified as belonging to a particular class of identifier, it must *stay* in that class for future versions of the standard. Because of that requirement for identifier stability, there are associated constraints on how the related character properties can change. In particular, the identifier-related properties listed in *Table 19a* may have their values for any particular assigned character change from No to Yes between versions of the standard, but once a character has the value Yes, that value is locked in, and cannot ever be changed back to No.

Table 19a. Yes-Locked Properties

Property Name	Abbr Name	Default Value
ID_Start	IDS	No
ID_Continue	IDC	No
XID_Start	XIDS	No
XID_Continue	XIDC	No

In some cases, a property is not immutable, but the list of possible values that it can have is considered invariant. For example, while at least some General_Category values are subject to change and correction, the enumerated set of possible values that the General_Category property can have is fixed and cannot be added to in the future. However, not all Enumeration properties used by Unicode algorithms have immutable lists of property values. For example, the enumerated lists of values associated with the Line_Break and the Word_Break properties have changed in the past, and may be changed again in future versions of the standard.

All characters other than those of General_Category Mn or Mc are guaranteed to have Canonical_Combining_Class=0.

In Unicode 4.0 and thereafter, the General_Category value *Decimal_Number* (Nd), and the Numeric_Type value *Decimal* (de) are defined to be co-extensive; that is, the set of characters having General_Category=Nd will always be the same as the set of characters having NumericType=de.

5.10.2 UCD File Format Invariants

There are also some constraints on allowable change in the file formats for UCD files. In general, the file format conventions are changed as little as possible, to minimize the impact on implementations which parse the machine-readable data files. However, some of the constraints on allowable file format change go beyond conservatism in format and instead have the status of invariants. These guarantees apply in particular to UnicodeData.txt, the very first data file associated with the UCD.

The number and order of the fields in UnicodeData.txt is fixed. Any additional information about character properties to be added to the UCD in the future will appear in separate data files, rather than being added as an additional field to UnicodeData.txt or by reinterpretation of any of the existing fields.

5.10.3 Invariants in Implementations

Applications may wish to take the various character property and file format invariants into account when choosing how to implement character properties.

The Canonical_Combining_Class offers a good example. The character property invariants regarding Canonical_Combining_Class guarantee that values, once assigned, will never change, and that all values used will be in the range 0..254. This means that the Canonical_Combining_Class can be safely implemented in an unsigned byte and that any value stored in a table for an existing character will not need to be updated dynamically for a later version.

In practice, for Canonical_Combining_Class far fewer than 256 values are used. Unicode 3.0 used 53 values; Unicode 3.1 through Unicode 4.1 used 54 values; and Unicode 5.0 through Unicode 9.0 used 55 values. New, non-zero Canonical_Combining_Class values are seldom added to the standard. (For details about this history, see DerivedCombiningClass.txt.) Implementations may take advantage of this fact for compression, because only the ordering of the non-zero values, and not their absolute values, matters for the Canonical Ordering Algorithm. In principle, it would be possible for up to 255 values to be used in the future, but the chances of the actual number of values exceeding 128 are remote at this point. There are implementation advantages in restricting the number of internal class values to 128—for example, the ability to use signed bytes without implicit widening to the int data type in Java.

5.11 Validation

The Unicode character property values in the UCD files can be validated by means of regular expressions. Such validation can also be useful in testing of implementations that return property values. The method of validation depends on the type of property, as described below. These expressions use Perl syntax, but may of course be converted to other formal conventions for use with other regular expression engines.

The regular expressions which are appropriate for validation of particular properties may change in each subsequent version of the UCD. However, because of stability guarantees for character property aliases, these regular expressions for one version of the Unicode Standard will match valid values for previous versions of the standard.

5.11.1 Enumerated and Binary Properties

Enumerated and binary character properties can be validated by generating a regular expression using the PropertyValueAliases.txt file. Because enumerated properties have a defined list of possible values, the validating regular expression simply ORs together all of the possible values. Binary properties are a special case of enumerated property, with a predefined very short list of possible values.

For example, to validate the East_Asian_Width property in the UCD, or to test an implementation that returns the East_Asian_Width property, parse the following relevant lines from PropertyValueAliases.txt and produce a regular expression that concatenates each of the short and

long property alias values.

```
# East_Asian_Width (ea)

ea ; A         ; Ambiguous
ea ; F         ; Fullwidth
ea ; H         ; Halfwidth
ea ; N         ; Neutral
ea ; Na        ; Narrow
ea ; W         ; Wide
```

The resulting regular expression would then be:

```
/A|Ambiguous|F|Fullwidth|H|Halfwidth|N|Neutral|Na|Narrow|W|Wide/
```

For each Unicode binary character property, the regular expression can be precomputed simply as:

```
/N|No|F|False|Y|Yes|T|True/
```

The Catalog properties, Age, Block, and Script, are another type of enumerated character property. All possible values of those properties for any given version of the Unicode Standard are listed in PropertyValueAliases.txt, so a validating regular expression for a Catalog property for that given version of the UCD can be generated by concatenating values, as for the other enumerated properties.

5.11.2 Canonical_Combining_Class Property

The Canonical_Combining_Class (ccc) property is a hybrid type. The possible values defined for it in UnicodeData.txt range from 0 to 254 and are numeric values. However, Canonical_Combining_Class also has symbolic aliases defined for those particular values that are in actual use; those symbolic aliases are listed in PropertyValueAliases.txt. To produce a validating regular expression for Canonical_Combining_Class, concatenate together the symbolic aliases from PropertyValueAliases.txt, and then add the numeric range 0..254.

The value 255 is reserved for use by implementations. When the ccc values are represented by bytes, that additional value of 255 may be used by an implementation for other purposes.

The value 133 is reserved. No characters have that value. The property value alias CCC133 is retained in accordance with the stability policy regarding property value aliases.

5.11.3 Unihan Properties

The validating regular expressions for each property tag defined in the Unihan database are described in detail in [UAX38].

5.11.4 Other Properties

Regular expressions to validate String and Miscellaneous properties in the UCD are provided in *Table 21*. Although Catalog properties may use strict tests, as described in *Section 5.11.1 Enumerated and Binary Properties*, generic patterns for Block and Script are also provided in *Table 21*.

To simplify the presentation of these expressions, commonly occurring subexpressions are first abstracted out as variables defined in *Table 20*.

Table 20. Common Subexpressions for Validation

Variable	Value	Notes and Examples
$digit	[0-9]	"0", "3"
$hexDigit	[0-9A-F]	"1", "A"
$alphaNum	[0-9A-Za-z]	"1", "A", "z"
$digits	$digit+	"0", "12345"
$label	$alphaNum+	"A", "Syriac", "NGKWAEN", "123467", "A005A"
$positiveDecimal	$digits\.$digits	"3.1"
$decimal	-?$positiveDecimal	"3.5", "-0.5"
$rational	-?$digits(/$digits)?	"3/4", "-3/4"
$optionalDecimal	-?$digits(\.$digits)?	"3.5", "-0.5", "2", "1000"
$name	$label((-\|- \|[-_])$label)*	name, with potential non-medial hyphens
$name2	$label([-_]$label)*	name, no non-medial hyphens allowed
$annotatedName	$name2(\(.*\))?	name with optional parenthetical annotation
$shortName	[A-Z]{0,3}	"", "O", "WA", "WAE"
$codePoint	(10|$hexDigit)?$hexDigit{4}	"00A0", "E0100", "10FFFF"
$codePoints	$codePoint(\s$codePoint)*	space-delimited list of 1 to n code points
$codePoint0	($codePoints)?	space-delimited list of 0 to n code points

The regular expressions listed in *Table 21* cover all the straightforward cases for other property values. For properties involving somewhat more irregular values, such as Age, ISO_Comment, and Unicode_1_Name, details for validation can be found in [UAX42].

Table 21. Regular Expressions for Other Property Values

Abbr	Name	Regex for Allowable Values	
nv	Numeric_Value	/$decimal/	Field 2
		/$optionalDecimal/	Field 3
		/$rational/	
blk	Block	/$name2/	
sc	Script		
dm	Decomposition_Mapping	/$codePoints/	
FC_NFKC	FC_NFKC_Closure		
NFKC_CF	NFKC_Casefold	/$codePoint0/	
cf	Case_Folding	/$codePoints/	
lc	Lowercase_Mapping		
tc	Titlecase_Mapping		
uc	Uppercase_Mapping		
scf	Simple_Case_Folding	/$codePoint/	
slc	Simple_Lowercase_Mapping		
stc	Simple_Titlecase_Mapping		
suc	Simple_Uppercase_Mapping		
bmg	Bidi_Mirroring_Glyph	/$codePoint/	
bpb	Bidi_Paired_Bracket	/$codePoint/	
EqUIdeo	Equivalent_Unified_Ideograph	/$codePoint/	
na	Name	/$name/	
Name_Alias	Name_Alias		
--	Names for named sequences*		
na1	Unicode_1_Name	/$annotatedName/	
JSN	Jamo_Short_Name	/$shortName/	

* The Unicode named character sequences constitute a string-valued property for an enumerated set of strings (the actual sequences which are given names). They follow the same syntax as the Name and Name_Alias property values and form part of the same namespace.

5.11.5 Validation of Multivalued Properties

Some properties, such as Script_Extensions of kCantonese, have property values each consisting of a set of element values. In the data files, these element values are separated by spaces. Validation of the property values is performed by first splitting each set into element values at the spaces, and then validating each element value individually. For example, the elements for Script_Extensions are values of the Script property; they are validated according to the validation requirements for the Script property. See also Section 5.7.6 Properties Whose Values Are Sets of Values.

The Name_Alias property has values which consist of sets of one or more name strings. In the data file for this property, each element value occurs on a separate line and can be validated as a separate element.

5.12 Deprecation

In the Unicode Standard, the term *deprecation* is used somewhat differently than it is in some other standards. Deprecation is used to mean that a character or other feature is strongly discouraged from use. This should not, however, be taken as indicating that anything has been removed from the standard, nor that anything is *planned* for removal from the standard. Any such change is constrained by the Unicode Consortium Stability Policies [Stability].

For the Unicode Character Database, there are two important types of deprecation to be noted. First, an *encoded character* may be deprecated. Second, a *character property* may be deprecated.

When an encoded character is strongly discouraged from use, it is given the property value Deprecated=True. The Deprecated property is a binary property defined specifically to carry this information about Unicode characters. Very few characters are ever formally deprecated this way; it is not enough that a character be uncommon, obsolete, disliked, or not preferred. Only those few characters which have been determined by the UTC to have serious architectural defects or which have been determined to cause significant implementation problems are ever deprecated. Even in the most severe cases, such as the deprecated format control characters (U+206A..U+206F), an encoded character is *never* removed from the standard. Furthermore, although deprecated characters are strongly discouraged from use, and should be avoided in favor of other, more appropriate mechanisms, they *may* occur in data. Conformant implementations of Unicode processes such a Unicode normalization *must* handle even deprecated characters correctly.

In the Unicode Character Database, a character property may also become strongly discouraged—usually because it no longer serves the purpose it was originally defined for. In such cases, the property is labelled "deprecated" in *Table 9, Property Table*. For example, see the Grapheme_Link property. Deprecated properties are not recommended for exposure in public APIs that support Unicode character properties.

5.13 Property APIs

The Unicode Standard does not specify the exact form of APIs which may be defined in software libraries to surface Unicode character properties to applications. However, there are some recommendations and general guidelines to follow, which should serve to reduce potential confusion and to promote better interoperability between applications using the Unicode Character Database.

In the discussion which follows here, the term *API* is used to refer to a particular function or method, whereas the term *API collection* is used to refer to a related group of APIs, which might constitute a set of functions exported from a library, a class definition, or other groupings of related functionality. A distinction is also made between a *public API*, which is exported for general application use, and a *private API*, which may be kept hidden within a library or class, intended for internal use.

First, if an API surfaces values of a particular Unicode character property and *purports* that value to represent a Unicode character property, it should exactly follow the specification of that property in the UCD. This principle follows from the general approach to conformance for the Unicode Standard: If you say it is Unicode, then it should follow the Unicode Standard specification.

Second, an API should be clear about which version of the UCD it supports. This can be done, for example, with documentation, either external or included in the source in header files, class definition notes, and so forth. For an API collection, an even better option is to include an API which explicitly reports which version of the UCD is supported. This provision should reduce confusion regarding particular property values which might change between versions of the Unicode Standard, as well as making it clear which repertoire of encoded characters is intended to be covered. There is no principled constraint on an API supporting *more than one* version of the UCD, as long as it is clear about how it does so.

Third, although there is no constraint on an API declaring that it only supports a designated subset of Unicode characters, best practice for a general purpose character property API would be to support the entire range of Unicode code points, providing determinant and well-documented property values for any valid Unicode code point input. That would include providing correct default property values for any unassigned code point. See *Section 2.2, Use of Default Values* for an explanation of that concept.

Fourth, a Unicode character property API is not precluded from extending or tailoring its support of character properties, as long as such behavior is clearly documented, so that applications understand the values they will be getting by calling the API. For example, an API might surface an extended new property such as IsDanda, which is not formally part of the properties specified by the UCD, but which can be inferred from the documentation of the Unicode Standard. An API supporting a particular tailoring of the Unicode Line Breaking Algorithm could surface tailored Line_Break property values to support that behavior. Alternatively, an API supporting a particular private use agreement could surface privately-defined properties for a designated range of PUA characters. All such use of APIs should be considered conformant ways of extending API collections using the UCD.

Designers of API collections to support Unicode character properties must also be aware that not all Unicode character properties are equal. There is no requirement, express or implied, that *all* Unicode character properties should be supported in a given API collection. In fact, an approach that simply parses the UCD and surfaces *all* Unicode character properties verbatim is very likely to result in a bad design. Character properties need to be understood in the context of the various Unicode algorithms they are designed to support.

The following subtypes of Unicode character properties should generally *not* be exposed in APIs, except in limited circumstances. They may not be useful, particularly in public API collections, and may instead prove misleading to the users of such API collections.

- *Contributory properties* are not recommended for public APIs.
- A subset of Unicode normalization-related properties are not recommended for public APIs. See *Section 5.7.5, Decompositions and Normalization*.
- Deprecated properties are not recommended for public APIs. See *Section 5.12, Deprecation*.

5.14 Character Age

The Age property indicates the first version in which a particular Unicode character was assigned. For example, U+20AC € EURO SIGN was added to Version 2.1 of the Unicode Standard, so it has age=2.1, while U+20B9 ₹ INDIAN RUPEE SIGN was added to Version 6.0 of the Unicode Standard, so it has age=6.0.

Formally, the Age property is a catalog property whose enumerated values correspond to a list of tuples consisting of a major version integer and a minor version integer. The major version is a positive integer constrained to the range 1..255. The minor version is a non-negative integer constrained to the range 0..255. These range limitations are specified so that implementations can be guaranteed that all valid, assigned Age values can be represented in a sequence of two unsigned bytes. A third value corresponding to the Unicode update version is not required, because new characters are never assigned in update versions of the standard.

The short values listed in PropertyValueAliases.txt for the Age property for assigned (designated) code points are of the form "m.n", with the first field corresponding to the major version, and the second field corresponding to the minor version.

The long values listed in PropertyValueAliases.txt for the Age property for assigned code points start with a "V" and use an underscore instead of a dot between the major and minor version numbers: V2_1, V6_0, and so on. This makes the long format more useful as an identifier in programming languages. It is also useful in regular expressions, where the dot has other significance.

The default value of the Age property, used for unassigned (undesignated) code points, is expressed with labels that depart from the numerical versioning scheme of the Age property for assigned code points; the short form for the default is "NA", and the long form for the default is "Unassigned". Implementations of parsers which manipulate the Age property need to be prepared for this special case, rather than expecting the default value to be expressed numerically, as "0.0", for example.

The Age property is based on when a character is encoded in the standard. It is normative and immutable, and cannot be meaningfully tailored.

The minimum value of the Age property is "1.1", instead of "1.0", because of the substantial and incompatible changes to the standard resulting from the merger of code points and character names between the Unicode Standard and ISO/IEC 10646 for their 1993 publications.

For Hangul syllable characters, which were extensively augmented in Unicode 2.0, the Age value is set to "2.0", even though a subset of the Hangul syllables had been published in earlier versions, at different code points.

Private use characters, noncharacter code points, and surrogate code points also get Age values. The private use characters and noncharacter code points on the BMP have age=1.1. However, the full architecture for UTF-16 and multiple planes was not fully documented until Unicode 2.0, so the private use characters and noncharacter code points on supplementary planes, as well as the surrogate code points in the range D800..DFFF, are given the value age=2.0.

The Age property cannot be derived from the other data files in any single version of the Unicode Character Database. Its derivation is done, rather, by tools that compare the assigned characters *between* subsequent versions. The data file DerivedAge.txt provides the definitive listing of the Age property value for all code points, as of that version of the standard.

The typical use case for the Age property in regular expressions is to search for all characters that were present in a given version. For this reason, an expression such as "\p{age=V3_0}" is exceptionally defined to match all of the code points assigned in Version 3.0—that is, all the code points with a value *less than or equal to* the value 3.0 for the Age property, rather than just the subset of those code points with the value 3.0. This interprets "\p{age=V3_0}" as the set of all characters assigned as of Unicode 3.0, rather than as just the set of characters *added* to Unicode 3.0 subsequent to the prior version. For more information, see Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18].

6 Test Files

The UCD contains a number of test data files. Those provide data in standard formats which can be used to test implementations of Unicode algorithms. The test data files distributed with this version of the UCD are listed in *Table 22*.

Table 22. Unicode Algorithm Test Data Files

File Name	Specification	Status	Unicode Algorithm
BidiTest.txt	[UAX9]	N	Unicode Bidirectional Algorithm
BidiCharacterTest.txt	[UAX9]	N	Unicode Bidirectional Algorithm
NormalizationTest.txt	[UAX15]	N	Unicode Normalization Algorithm
LineBreakTest.txt	[UAX14]	N	Unicode Line Breaking Algorithm
GraphemeBreakTest.txt	[UAX29]	N	Grapheme Cluster Boundary Determination
WordBreakTest.txt	[UAX29]	N	Word Boundary Determination
SentenceBreakTest.txt	[UAX29]	N	Sentence Boundary Determination

The normative status of these test files reflects their use to determine the correctness of implementations claiming conformance to the respective algorithms listed in the table. There is no requirement that any particular Unicode implementation also implement the Unicode Line Breaking Algorithm, for example, but *if* it implements that algorithm correctly, it should be able to replicate the test case results specified in the data entries in LineBreakTest.txt.

6.1 NormalizationTest.txt

This file contains data which can be used to test an implementation of the Unicode Normalization Algorithm. (See [UAX15] and [Tests15].)

The data file has a Unicode string in the first field (which may consist of just a single code point). The next four fields then specify the expected output results of converting that string to Unicode Normalization Forms NFC, NFD, NFKC, and NFKD, respectively. There are many tricky edge cases included in the input data, to ensure that implementations have correctly implemented some of the more complex subtleties of the Unicode Normalization Algorithm.

The header section of NormalizationTest.txt provides additional information regarding the normalization invariant relations that any conformant implementation should be able to replicate.

The Unicode Normalization Algorithm is not tailorable. Conformant implementations should be expected to produce results as specified in NormalizationTest.txt and should not deviate from those results.

6.2 Segmentation Test Files and Documentation

LineBreakTest.txt, located in the auxiliary directory of the UCD, contains data which can be used to test an implementation of the Unicode Line Breaking Algorithm. (See [UAX14] and [Tests14].) The header of that file specifies the data format and the use of the test data to specify line break opportunities. Note that non-ASCII characters are used in this test data as field delimiters.

There is an associated documentation file, LineBreakTest.html, which displays the results of the Line Breaking Algorithm in an interactive chart form, with a documented listing of the rules.

The Unicode text segmentation test data files are also located in the auxiliary directory of the UCD. (See [Tests29].) They contain data which can be used to test an implementation of the segmentation algorithms specified in [UAX29]. The headers of those file specify the data format and the use of the test data to specify text segmentation opportunities. Note that non-ASCII characters are used in this test data as field delimiters.

There are also associated documentation files, which display the results of the segmentation algorithms in an interactive chart form, with a documented listing of the rules:

- GraphemeBreakTest.html
- SentenceBreakTest.html
- WordBreakTest.html

Unlike the Unicode Normalization Algorithm, the Unicode Line Breaking Algorithm and the various text segmentation algorithms are tailorable, and there is every expectation that implementations will tailor these algorithms to produce results as needed. The test data files only test the *default* behavior of the algorithms. Testing of tailored implementations will need to modify and/or extend the test cases as appropriate to match any documented tailoring.

6.3 Bidirectional Test Files

These files contain data which can be used to test an implementation of the Unicode Bidirectional Algorithm. (See [UAX9] and [Tests9].)

The data in BidiTest.txt is intended to exhaustively test all possible combinations of Bidi_Class values for strings of length four or less. To allow for the resulting very large number of test cases, the data file has a somewhat complicated format which is described in the header. Fundamentally, for each input string and for each possible input paragraph level, the test data specifies the resulting bidi levels and expected reordering.

The data in BidiCharacterTest.txt is provided to test various edge cases for the algorithm. It contains an extra field which allows for explicit control of the overall directional context for each test case.

The Unicode Bidirectional Algorithm is tailorable within certain limits. Conformant implementations with no tailoring are expected to produce the results as specified in BidiTest.txt and BidiCharacterTest.txt, and should not deviate from those results. Tailored implementations can also use the data in the test files to test for overall conformance to the algorithm by changing the assignment of properties to characters to reflect the details of their tailoring.

7 UCD Change History

This section summarizes the recent changes to the UCD—including its documentation files—and is organized by Unicode versions.

References in the change history are ~~often~~ sometimes made to a Public Review Issue (PRI). See https://www.unicode.org/review/resolved.html for more information about each of those cases.

Unicode 16.0.0

Changes in specific files:

Appropriate existing data files were updated to add the NNNN new characters encoded in Unicode 16.0. Major changes that are most likely to affect implementations are documented in Section M of the Unicode 16.0.0 page. Significant data file updates resulting from encoding the new characters and from various character property changes are summarized below, in the same grouping manner used in Components of Unicode 16.0.0.

Note that minor editorial updates and changes to the derived and extracted data files are not documented here. Routine additions of expected property values for newly encoded characters are likewise not called out explicitly in this summary.

Core Data

- Blocks.txt
  - TBD
- CaseFolding.txt
  - TBD
- CJKRadicals.txt
  - TBD
- DerivedCoreProperties.txt
  - TBD
- DerivedNormalizationProps.txt
  - TBD
- LineBreak.txt
  - TBD
- NamesList.txt
  - Content was updated throughout with new characters, as well as annotations, cross references, subheadings, and new comments.
- PropertyAliases.txt
  - TBD
- PropertyValueAliases.txt
  - The 16.0 value, with the alias V16_0, was added to the catalog property Age.
  - TBD
- PropList.txt
  - TBD
- ScriptExtensions.txt
  - TBD

Unihan Database (Unihan.zip)

- Unihan_DictionaryIndices.txt
  - TBD
- Unihan_DictionaryLikeData.txt
  - TBD
- Unihan_IRGSources.txt
  - TBD

- Unihan_NumericValues.txt
  - TBD
- Unihan_OtherMappings.txt
  - TBD
- Unihan_RadicalStrokecounts.txt
  - TBD
- Unihan_Readings.txt
  - TBD
- Unihan_Variants.txt
  - TBD

Data for UAX #45

- USourceData.txt
  - TBD
- USourceGlyphs.pdf
  - Glyphs were added for the NN new UTC-Source ideographs introduced in USourceData.txt.
- USourceRSChart.pdf
  - Added new entries for the radical-stroke index.

Extracted Data

No specific items to highlight.

Conformance Test Data

No specific items to highlight.

Auxiliary Data for UAX #14 and UAX #29

- SentenceBreakProperty.txt
  - TBD
- WordBreakProperty.txt
  - TBD
- LineBreakTest.txt
  - TBD

Documentation for Auxiliary Data

No specific items to highlight.

Emoji Data

No specific items to highlight.

Unicode 15.1.0

Changes in specific files:

Appropriate existing data files were updated to add the 627 new characters encoded in Unicode 15.1. Major changes that are most likely to affect implementations are documented in Section M of the Unicode 15.1.0 page. Significant data file updates resulting from encoding the new characters and from various character property changes are summarized below, in the same grouping manner used in Components of Unicode 15.1.0.

Note that minor editorial updates and changes to the derived and extracted data files are not documented here. Routine additions of expected property values for newly encoded characters are likewise not called out explicitly in this summary.

Core Data

- Blocks.txt
  - Added CJK Extension I block.
- CaseFolding.txt
  - Added S(imple casefolding) entries for 1FD3, 1FE3, and FB05.
- CJKRadicals.txt
  - Removed entry for 162'.
  - Added several entries for radical variants using the two apostrophes convention.
- DerivedCoreProperties.txt
  - Added the new derived property Indic_Conjunct_Break (InCB).
- DerivedNormalizationProps.txt
  - Added the new derived property NFKC_Simple_Casefold (NFKC_SCF).
- EastAsianWidth.txt

- Spaces are now allowed around the semicolon field delimiter in this file, to provide better formatting and for consistency with most other UCD data files.

- LineBreak.txt
  - New Line_Break classes were added for a significant number of Indic scripts, to match updates to UAX #14 for support of line breaking at orthographic syllable boundaries.
  - Spaces are now allowed around the semicolon field delimiter in this file, to provide better formatting and for consistency with most other UCD data files.
- NamesList.txt
  - Content was updated throughout with new characters, as well as annotations, cross references, subheadings, and new comments.
- PropertyAliases.txt
  - New property aliases were added: IDSU, ID_Compat_Math_Start, ID_Compat_Math_Continue, InCB, and NFKC_SCF.
- PropertyValueAliases.txt
  - The 15.1 value, with the alias V15_1, was added to the catalog property Age.
  - The CJK_Ext_I value alias was added to the Block property.
  - Five new Line_Break property value aliases were added: AK, AP, AS, VF, VI.
  - Aliases were added for the new values for InCB.
  - The new binary properties have the same default aliases as for other binary properties.
- PropList.txt
  - 200C, 200D, 30FB, and FF65 were added to the contributory property Other_ID_Continue, to simplify a derivation for identifier-related properties.
  - Values were added for ID_Compat_Math_Start and ID_Compat_Math_Continue.
  - 17D4 and 17D5 were added to Sentence_Terminal.
  - The additional ideographic description characters were given the expected IDS_Binary_Operator property values, but two new ideographic description characters got the new IDS_Unary_Operator property value.
- ScriptExtensions.txt
  - Several Vedic marks and North Indic numeric characters were added to large sets shared by a number of Indic scripts.

Unihan Database (Unihan.zip)

- Unihan_DictionaryIndices.txt
  - Added the provisional kSMSZD2003Index property with approximately 11,000 records.
  - Added approximately 35,000 records to the provisional kMorohashi property.
  - Changed approximately 4,000 provisional kMorohashi property values.
  - Removed the provisional kIRGDaiKanwaZiten property and its records.
- Unihan_DictionaryLikeData.txt
  - Added the provisional kMojiJoho property with approximately 53,000 records.
  - Added approximately 650 records to the provisional kFourCornerCode property.
  - Added one record to the provisional kPhonetic property.
  - Added approximately 60 records to the provisional kStrange property.
  - Changed seven provisional kStrange property values.
- Unihan_IRGSources.txt
  - Added kIRG_GSource, kRSUnicode, and kTotalStrokes records for the characters in the new CJK Unified Ideographs Extension I block, which included the new "GIDC23-" prefix.
  - Added three new records to the kIRG_KPSource property.
  - Added four new records to the kIRG_TSource property.
  - Added two new records to the kIRG_USource property.
  - Added six new records to the kIRG_VSource property.
  - Changed one kIRG_GSource property value.
  - Changed four kIRG_KPSource property values.
  - Changed two kIRG_VSource property values.
  - Changed approximately 225 kRSUnicode property values.
  - Changed approximately 275 kTotalStrokes property values.
  - Removed three records from the kIRG_KPSource property.
  - Removed one record from the kIRG_USource property.
- Unihan_NumericValues.txt
  - Added the provisional kVietnameseNumeric property with 50 records.
  - Added the provisional kZhuangNumeric property with 13 records.
  - Added eight new records to the provisional kOtherNumeric property.
  - Added three new records to the provisional kPrimaryNumeric property.
  - Changed one provisional kPrimaryNumeric property value.
  - Removed one record from the provisional kAccountingNumeric property.
- Unihan_OtherMappings.txt
  - Added two new records to the provisional kBigFive property.
  - Removed the provisional kHKSCS, kKPS0, kKPS1, kKSC0, and kKSC1 properties and their records.
- Unihan_RadicalStrokecounts.txt
  - Removed the provisional kRSKangXi property and its records.
- Unihan_Readings.txt
  - Added the provisional kJapanese property with approximately 52,000 records.

- Added the provisional kSMSZD2003Readings property with approximately 8,000 records.
- Added approximately 125 new records to the provisional kCantonese property.
- Added approximately 350 new records to the provisional kDefinition property.
- Added approximately 250 new records to the provisional kXHC1983 property.
- Changed approximately 20 provisional kCantonese property values.
- Changed approximately 150 provisional kDefinition property values.
- Changed approximately 150 provisional kXHC1983 property values.
- Removed one record from the provisional kDefinition property.
- Removed approximately 200 records from the provisional kXHC1983 property.

- Unihan_Variants.txt
  - Added approximately 15 new records to the provisional kSemanticVariant property.
  - Added approximately 50 new records to the provisional kSimplifiedVariant property.
  - Added six new records to the provisional kSpecializedSemanticVariant property.
  - Changed approximately 125 provisional kSemanticVariant property values.
  - Changed 11 provisional kSimplifiedVariant property values.
  - Changed two provisional kSpecializedSemanticVariant property values.
  - Changed approximately 60 provisional kTraditionalVariant property values.
  - Removed two records from the provisional kTraditionalVariant property.

Data for UAX #45

- USourceData.txt
  - 39 new records were added for new UTC-Source ideographs.
  - The "ExtI" (Extension I) status value was added.
  - The single-letter status values "N," "V," "W," and "X" were changed to the more descriptive "FutureWS," "Variant," "Rejected," and "NoAction" status values.
  - The status values of various records were updated so that the "UK-2015" and "WS-2017" status values could be removed.
  - Various records were updated to add or improve their ideographic description sequences.
- USourceGlyphs.pdf
  - Glyphs were added for the 39 new UTC-Source ideographs introduced in USourceData.txt.
- USourceRSChart.pdf
  - Added new entries for the radical-stroke index.

Extracted Data

No specific items to highlight.

Conformance Test Data

No specific items to highlight.

Auxiliary Data for UAX #14 and UAX #29

- SentenceBreakProperty.txt
  - A number of prepended concatenation marks changed from sb=Format to sb=Numeric, for better break boundaries when used with numbers.
- WordBreakProperty.txt
  - A number of prepended concatenation marks changed from wb=Format to wb=Numeric, for better break boundaries when used with numbers.
- LineBreakTest.txt
  - The line breaking test cases were extended significantly, to deal with the extensions of the LB algorithm to deal with orthographic syllable breaks.

Documentation for Auxiliary Data

No specific items to highlight.

Emoji Data

No specific items to highlight.

~~**Unicode 15.0.0**~~

~~**Changes in specific files:**~~

~~Appropriate existing data files were updated to add the 4489 new characters encoded in Unicode 15.0. Major changes that are most likely to affect implementations are documented in Section M of the Unicode 15.0.0 page. Significant data file updates resulting from encoding the new characters and from various character property changes are summarized below, in the same grouping manner used in Components of Unicode 15.0.0.~~

~~Note that minor editorial updates and changes to the derived and extracted data files are not documented here. Routine additions of expected property values for newly encoded characters are likewise not called out explicitly in this summary.~~

Core Data

- Blocks.txt
  - Seven new blocks were added, six allocated in the Supplementary Multilingual Plane, and one in the Tertiary Ideographic Plane. These include two blocks for the newly encoded scripts in Version 15.0 – Kawi and Nag Mundari.
  - The block range for Egyptian Hieroglyph Format Controls was extended to end at 1345F, instead of 1343F.
- IndicPositionalCategory.txt
  - 0953 and 0954 were removed from InPC=Top.
  - Five Kayah Li vowel signs, A926..A92A, were added to InPC=Top.
- IndicSyllabicCategory.txt
  - Clarification was added re 0A71 GURMUKHI ADDAK.
  - 0AFB GUJARATI SIGN SHADDA was changed from InSC=Cantillation_Mark to InSC=Gemination_Mark.
  - The note regarding the Brahmi Joining Number class was expanded.
- LineBreak.txt
  - The double diacritic marks 1DCD and 1DFC were changed from lb=CM to lb=GL.
  - 2057 QUADRUPLE PRIME was changed from lb=AL to lb=PO.
- NamedAliases.txt
  - One formal name alias of type abbreviation was added for 0019.
  - Two formal name aliases of type correction were added for 0616 and 1BBD.
- NamesList.txt
  - Content was updated throughout with new characters, as well as annotations, cross references, subheadings, and new comments.
- PropertyValueAliases.txt
  - The 150 value, with the alias V150, was added to the catalog property Age.
  - Aliases were added for new Script and new Block property values.
  - Redundant @missing lines were removed for Bidi_Mirroring_Glyph, Equivalent_Unified_Ideograph, NFKC_Casefold, and Script_Extensions.
- PropList.txt
  - Two Tibetan bindus, 0F82..0F83, were added to Other_Alphabetic.
  - Several overlooked modifier letters, 10FC, A7F2..A7F4, and AB69, were added to Other_Lowercase, for consistency with other modifier letters.
- StandardizedVariants.txt
  - 94 standardized variation sequences were added for rotational variants of Egyptian hieroglyph signs.
  - 4 standardized variation sequences were added for expanded variants of Egyptian hieroglyph lost signs.

Unihan Database (Unihan.zip)

- Unihan_DictionaryIndices.txt
  - Added a single new kCheungBauerIndex record.
  - Added the approximately 14,000 records for the kCihaiT property (moved from Unihan_DictionaryLikeData.txt).
  - Changed two kDaeJaweon property values.
  - Added approximately 50,000 new records to the kKangXi property that were derived from the kIRG_GSource and kIRGKangXi properties, changed approximately 250 kKangXi property values, and removed approximately 30 records with meaningless property values.
  - Removed approximately 300 records for the kMorohashi property with meaningless property values.
- Unihan_DictionaryLikeData.txt
  - Added the kAlternateTotalStrokes property with approximately 100 records.
  - Removed the approximately 14,000 records for the kCihaiT property (moved to Unihan_DictionaryIndices.txt).
  - Changed a very small number of kHKGlyph property values.
  - Removed approximately 400 records for the kPhonetic property.
- Unihan_IRGSources.txt
  - Added the "GXM-" and "GZA-" prefixes to the kIRG_GSource property, along with new records.
  - Added the "T12-" prefix to the kIRG_TSource property, along with new records.
  - Added new records to the kIRG_HSource, kIRG_KSource, and kIRG_USource properties.
  - Changed a single kIRG_VSource property value as a result of a disunification.
  - Changed a very small number of kRSUnicode and kTotalStrokes property values.
  - Moved a single kIRG_UKSource source reference.
  - Added IRG source data, kRSUnicode, and kTotalStrokes records for a single new character that was appended to the CJK Unified Ideographs Extension G block.
  - Added IRG source data, kRSUnicode, and kTotalStrokes records for the characters in the newly encoded CJK Unified Ideographs Extension H block.
- Unihan_RadicalStrokeCounts.txt
  - Changed two kRSKangXi property values.
- Unihan_Readings.txt
  - Changed approximately 150 kCantonese property values, and added approximately 150 new kCantonese records.
  - Changed approximately 800 kDefinition property values, and added approximately 1500 new kDefinition records.
  - Changed a single kVietnamese property value, and added a very small number of new kVietnamese records.
- Unihan_Variants.txt
  - Changed a small number of kSemanticVariant property values, and added approximately 100 new kSemanticVariant records.

- ~~Changed a very small number of kSimplifiedVariant property values, and added approximately 400 new kSimplifiedVariant records.~~
- ~~Changed a small number of kSpecializedSemanticVariant property values, and added an even smaller number of new kSpecializedSemanticVariant records.~~
- ~~Removed a very small number of kSpoofingVariant records.~~
- ~~Changed approximately 400 kTraditionalVariant property values, and added a small number of new kTraditionalVariant records.~~
- ~~Added a very small number of new kZVariant records, and removed an even smaller number of kZVariant records.~~

~~Data for UAX #45~~

- ~~USourceData.txt~~
  - ~~59 new entries were added for new UTC-Source ideographs.~~
  - ~~Various entries were updated to reflect their encoding as part of CJK Extension H.~~
  - ~~Status field values for various CJK Extensions were all changed from single letter values to four letter values ("A" → "ExtA", etc.), as documented in UAX #45.~~
- ~~USourceGlyphs.pdf~~
  - ~~Glyphs were added for the 59 new UTC-Source ideographs introduced in USourceData.txt.~~
- ~~USourceRSChart.pdf~~
  - ~~Added new entries for the radical-stroke index.~~

~~Extracted Data~~

- ~~DerivedBidiClass.txt~~
  - ~~Multiple @missing lines were added, to deal with all default value range assignments.~~
  - ~~Values explicitly assigned formerly to unassigned code points were removed, because they are now redundant.~~
- ~~DerivedLineBreak.txt~~
  - ~~Multiple @missing lines were added, to deal with all default value range assignments.~~
  - ~~Values explicitly assigned formerly to unassigned code points were removed, because they are now redundant.~~

~~Conformance Test Data~~

~~No specific items to highlight.~~

~~Auxiliary Data for UAX #14 and UAX #29~~

- ~~SentenceBreakProperty.txt~~
  - ~~Several modifier letters (10FC, A7F2..A7F4, and AB69) changed from OLetter to Lower, as a result of their addition to Other_Lowercase.~~

~~Documentation for Auxiliary Data~~

~~No specific items to highlight.~~

~~Emoji Data~~

~~No specific items to highlight.~~

Acknowledgments

Mark Davis and Ken Whistler are the authors of the initial version and have added to and maintained the text of this annex. Laurențiu Iancu assisted in the documentation of UCD changes for Versions 6.3.0 through 13.0.0. Ken Lunde and John Jenkins assisted in the documentation of Unihan changes for Versions 13.0.0 through 15.0.0, and Ken Lunde continued this work for Version 15.1.0. Julie Allen and Asmus Freytag provided editorial suggestions for improvement of the text. Over the years, many members of the UTC have participated in the review of the UCD and its documentation.

References

For references for this annex, see Unicode Standard Annex #41, "Common References for Unicode Standard Annexes."

Modifications

The following summarizes modifications from previous revisions of this annex.

Revision 33 [KW]

- **Proposed Update** for Unicode 16.0.0.
- Added documentation of new property Modifier_Combining_Mark.
- Clarified that the Numeric_Value of a Han character is set based on the *first* value in the kPrimaryNumeric, kAccountingNumeric, or kOtherNumeric tag, as those tags may include comma-separated lists of values.
- Removed an obsolete statement regarding the restriction of spaces around semicolons in Section 4.2.1, Data Fields.
- Took note of the minor format change for EastAsianWidth.txt and LineBreak.txt in the 15.1.0 UCD change record.

~~Revision 32 [KW]~~

- **Proposed Update** for Unicode 15.1.0.
- Corrected name of property to Canonical_Combining_Class in Section 5.11.2.
- Changed Bidi_Mirroring_Glyph, Bidi_Paired_Bracket, and Equivalent_Unified_Ideograph from type "M" to type "S" (string-valued) in the Property Table.
- Added clarification in Section 4.2.9 that some string-valued properties have <none> as their default value.
- In Section 4.2.10 changed the interpretation of <none> as a default value from "the empty string" to "no value is defined".
- Added IDS_Unary_Operator to the Property Table.
- Added NFKC_Simple_Casefold to the Property Table.
- Added ID_Compat_Math_Continue and ID_Compat_Math_Start to the Property Table.
- Added Indic_Conjunct_Break to the Property Table.
- Added clarification in Section 4.1.4 that file names no longer contain version and draft suffixes during alpha and beta review, and that the location of draft files under review has been changed.
- Updated the outdated kCantonese example in Section 5.7.6 with a corrected kVietnamese example.
- Small editorial corrections passim.

Modifications for previous versions are listed in those respective versions.
