Technical Reports **L2/10-176**

Proposed Update

Unicode Standard Annex #44

UNICODE CHARACTER DATABASE

Version	Unicode 6.0.0 draft 10
~~Authors~~ Editors	Mark Davis (markdavis@google.com) and Ken Whistler (ken@unicode.org)
Date	2010-05-07
This Version	http://www.unicode.org/reports/tr44/tr44-5.html
Previous Version	http://www.unicode.org/reports/tr44/tr44-4.html
Latest Version	http://www.unicode.org/reports/tr44/
Latest Proposed Update	http://www.unicode.org/reports/tr44/proposed.html
Revision	5

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*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time.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 ~~5.2.0~~ 6.0.0 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].

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

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

The specific files for the UCD associated with this version of the Unicode Standard (6.0.0) are located at:

http://www.unicode.org/Public/6.0.0/

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

http://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*, 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

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.

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.

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.

2.2 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 Web site, 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.

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. ~~For example, the Grapheme_Link property is deprecated.~~ 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

2.3.3 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 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. ~~For example, the Hyphen property~~

~~was stabilized as of Version 4.0.0.~~

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 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 printing of the Unicode code charts and names list. See also *Section 17.1, Character Names List*, in [Unicode] for a detailed discussion of the conventions used in the Unicode names list as formatted for printing.

3.4 StandardizedVariants.html

StandardizedVariants.html documents standardized variants, showing a representative glyph for each. It is closely tied to the data file, StandardizedVariants.txt, which defines those sequences normatively.

3.5 Unihan and UAX #38

Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38] describes the format and content of the Unihan Database, 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.

3.6 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, it is the list which is to be taken as normative. See Simple and Derived Properties.

3.7 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, focussed documentation files, UnicodeCharacterDatabase.html, PropList.html, and DerivedProperties.html, which were later consolidated into UCD.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 Web site. 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 (6.0.0) are located at:

http://www.unicode.org/Public/6.0.0/

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. In Version 6.0.0 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.

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 for any particular version of the UCD, a beta review is conducted. The beta review files are located in the same directory that is later used for the released UCD, but during the beta review period, the subdirectory structure differs

somewhat and may contain temporary files, including documentation of diffs between deltas for the beta review. Also, during the beta review, all data file names are suffixed with version numbers and delta numbers. So a typical file name during beta review may be "PropList-5.2.0d13.txt" instead of the finally released "PropList.txt".

Notices contained in a ReadMe.txt file in the 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 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.
- The Unihan data files 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.

4.2.2 Code Points and Sequences

- Code points are expressed as hexadecimal numbers with four to six digits. 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;;;;;
9FC3;<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—Normative* 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 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:

```
00BC..00BE ; numeric # No [3] VULGAR FRACTION ONE QUARTER..VULGAR FRACTION TH
```

- 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.

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..2069    ; Default_Ignorable_Code_Point # Cn   [5] <reserved-2065>..<res
```

- Code point labels use one of the tags as documented in *Section 4.8, Name—Normative* 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.

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 Values

- 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 SYMBO
```

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
180E       ; White_Space # Zs      MONGOLIAN VOWEL SEPARATOR
2000..200A ; White_Space # Zs [11] EN QUAD..HAIR SPACE
```

4.2.8 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 string properties, including the definition of foldings, the default value is the code point of the character itself.
- For miscellaneous properties which take strings as values, such as the Unicode Name property, the default value is a null string.
- For binary properties, 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 *Table 4*.
- Default values may also be listed in specially formatted comment lines, using the keyword "@missing". For example:

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

- 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 the Default Values for Properties table below if they cannot be derived from the general rules about default values for properties.

Default values for common catalog, enumeration, and numeric properties are listed in *Table 4*.

Table 4. Default Values for Properties

Property Name	Default Value
Age	unassigned
Bidi_Class	L, AL, R
Block	No_Block
Canonical_Combining_Class	Not_Reordered (= 0)
Decomposition_Type	None
East_Asian_Width	Neutral (= N), Wide (= W)
General_Category	Cn
Numeric_Type	None
Numeric_Value	NaN
Script	Unknown (= Zzzz)

Default values for the Unicode character property Bidi_Class are complex. See Unicode Standard Annex #9, "The Unicode Bidirectional Algorithm" [UAX9] and DerivedBidiClass.txt for more details.

Default values for the East_Asian_Width property are also complex. 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 DerivedEastAsianWidth.txt for more details.

4.2.9 Text Encoding

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

4.2.10 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.

4.2.11 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
```

4.2.12 Other File Formats

- The data format for Unihan data files 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, 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 Web site 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 Table 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 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,

Informative, or **P**rovisional.

Table 5. Files in the UCD

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.
Index.txt	Chapter 17	I	Index to Unicode characters, as printed in the Unicode Standard.
NamesList.txt	Chapter 17	I	Names list used for production of the code charts, derived from UnicodeData.txt. It contains additional annotations.
NamesList.html	Chapter 17	I	Documents the format of NamesList.txt.
StandardizedVariants.txt	Chapter 16	N	Lists all the standardized variant sequences that have been defined, plus a textual description of their desired appearance.
StandardizedVariants.html	Chapter 16	N	A derived documentation file, generated from StandardizedVariants.txt, plus a list of sample glyphs showing the desired appearance of each standardized variant.
NamedSequences.txt	[UAX34]	N	Lists the names for all approved named sequences.
NamedSequencesProv.txt	[UAX34]	P	Lists the names for all provisional named sequences.

For more information about these files and their use, see the referenced annexes or chapters of Unicode Standard.

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 Web site. 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.

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.

~~An index of all the non-CJK character properties by name can be found below in the Property Summary section.~~

5.1 Property ~~Summary~~ Index

Table 7 provides a summary list of the Unicode character properties, excluding most of those specific to the Unihan data files. 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. The link on each property leads to its description in ~~the~~ *Table 9, Property Table* ~~above~~.

Table 7. Property ~~Summary Table~~ Index by Scope of Use

General	Normalization	CJK
Name	Canonical_Combining_Class	Ideographic
Name_Alias	Decomposition_Mapping	Unified_Ideograph
Block	Composition_Exclusion	Radical
Age	Full_Composition_Exclusion	IDS_Binary_Operator
General_Category	Decomposition_Type	IDS_Trinary_Operator
Script	FC_NFKC_Closure (deprecated)	Unicode_Radical_Strok
White_Space	NFC_Quick_Check	**Miscellaneous**
Alphabetic	NFKC_Quick_Check	Math
Hangul_Syllable_Type	NFD_Quick_Check	Quotation_Mark
Noncharacter_Code_Point	NFKD_Quick_Check	Dash
Default_Ignorable_Code_Point	Expands_On_NFC (deprecated)	Hyphen (deprecated,
Deprecated	Expands_On_NFD (deprecated)	STerm
Logical_Order_Exception	Expands_On_NFKC (deprecated)	Terminal_Punctuation
Variation_Selector	Expands_On_NFKD (deprecated)	Diacritic
	NFKC_Casefold	Extender
	Changes_When_NFKC_Casefolded	Grapheme_Base
Case	**Shaping and Rendering**	Grapheme_Extend
Uppercase	Join_Control	Grapheme_Link (depre
Lowercase	Joining_Group	Unicode_1_Name
Lowercase_Mapping	Joining_Type	ISO_Comment (~~obsole~~ stabilized)
Titlecase_Mapping	Line_Break	
Uppercase_Mapping	Grapheme_Cluster_Break	
Case_Folding	Sentence_Break	**Contributory Proper**
Simple_Lowercase_Mapping	Word_Break	Other_Alphabetic
Simple_Titlecase_Mapping	East_Asian_Width	Other_Default_Ignorab
Simple_Uppercase_Mapping	**Bidirectional**	Other_Grapheme_Exte
Simple_Case_Folding	Bidi_Class	Other_ID_Start
Soft_Dotted	Bidi_Control	Other_ID_Continue
Cased	Bidi_Mirrored	Other_Lowercase
Case_Ignorable	Bidi_Mirroring_Glyph	Other_Math

Changes_When_Lowercased	**Identifiers**	Other_Uppercase
Changes_When_Uppercased	ID_Continue	Jamo_Short_Name
Changes_When_Titlecased	ID_Start	**Numeric**
Changes_When_Casefolded	XID_Continue	Numeric_Value
Changes_When_Casemapped	XID_Start	Numeric_Type
	Pattern_Syntax	Hex_Digit
	Pattern_White_Space	ASCII_Hex_Digit

5.2 About the Property Table

Table 9, Property Table ~~The big property table below, *Table 8,*~~ specifies the list of character properties defined in the UCD. ~~*Table 8*~~ 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, and UnicodeData.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	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** 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.

- **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, and the Unicode_Radical_Stroke property (a combination of numeric values) documented in Unicode Standard Annex #38, "Unicode Han Database (Unihan)" [UAX38].

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

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 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. Property Table

ArabicShaping.txt			
Joining_Type Joining_Group	E	N	Basic Arabic and Syriac character shaping p initial, medial and final shapes. See *Section* [Unicode].
BidiMirroring.txt			
Bidi_Mirroring_Glyph	S	I	Informative mapping for substituting chara implementation of bidirectional mirroring. T characters with the Bidi_Mirrored property t that normally are displayed with the corresp glyph. When a character with the Bidi_Mirro default value for Bidi_Mirroring_Glyph, that

			character exists whose glyph is appropriate glyph mirroring. Implementations must the mechanisms to implement mirroring of tho Unicode Bidirectional Algorithm. See Unicod #9:" The Unicode Bidirectional Algorithm" [confuse this property with the Bidi_Mirrore
Blocks.txt			
Block	C	N	List of block names, which are arbitrary nar code points. See Chapter 17 the code chart
CompositionExclusions.txt			
Composition_Exclusion	B	N	~~Properties for~~ A property used in normaliza Standard Annex #15: "Unicode Normalizatio Unlike other files, CompositionExclusions.t relevant code points.
CaseFolding.txt			
Simple_Case_Folding Case_Folding	S	N	Mapping from characters to their case-fold informative file containing normative derive *Derived from UnicodeData and SpecialCasin* **Note:** The case foldings are omitted in the the same as the code point itself.
DerivedAge.txt			
Age	C	N/I	This file shows when various code points w designated/assigned in successive versions Standard. The Age property is normative in the sense specified based on when a character is enco However, DerivedAge.txt is provided for inf of the Age property for a code point can be of successive versions of the UCD, and Age normatively in the specification of any Unic **Note:** When using the Age property in regu expression such as "\p{age=3.0}" matches a assigned in Version 3.0—that is, all the cod *less than* or equal to 3.0 for the Age proper information, see Unicode Technical Standar Regular Expressions" [UTS18].
EastAsianWidth.txt			
East_Asian_Width	E	I	Properties for determining the choice of wi glyphs in East Asian contexts. Property valu Unicode Standard Annex #11, "East Asian W
HangulSyllableType.txt			
Hangul_Syllable_Type	E	N	The values L, V, T, LV, and LVT used in *Cha* in [Unicode].
Jamo.txt			

Jamo_Short_Name	M	C	The Hangul Syllable names are derived from Names, as described in *Chapter 3, Conform*
LineBreak.txt			
Line_Break	E	N	Properties for line breaking. For more infor Standard Annex #14, "Unicode Line Breakin [UAX14].
GraphemeBreakProperty.txt			
Grapheme_Cluster_Break	E	I	See Unicode Standard Annex #29, "Unicode [UAX29]
SentenceBreakProperty.txt			
Sentence_Break	E	I	See Unicode Standard Annex #29, "Unicode [UAX29]
WordBreakProperty.txt			
Word_Break	E	I	See Unicode Standard Annex #29, "Unicode [UAX29]
NameAliases.txt			
Name_Alias	M	N	Normative formal aliases for characters with as described in *Chapter 4, Character Proper* These aliases exactly match the formal alias Unicode Standard code charts.
NormalizationCorrections.txt			
used in Decomposition Mappings	S	N	NormalizationCorrections lists code point d *Normalization Corrigenda*. For more inform Standard Annex #15, "Unicode Normalizati
Scripts.txt			
Script	C	I	Script values for use in regular expressions. information, see Unicode Standard Annex # Property" [UAX24].
SpecialCasing.txt			
Uppercase_Mapping Lowercase_Mapping Titlecase_Mapping	S	I	Data for producing (in combination with th mappings from UnicodeData.txt) the full ca
Unihan data files (for more information, see [UAX38])			
Numeric_Type Numeric_Value	E	I	The characters tagged with either kPrimaryN kAccountingNumeric, or kOtherNumeric are value Numeric_Type=Numeric, and the Num in those tags. Most characters have these numeric propert from UnicodeData.txt. See Numeric_Type.
Unicode_Radical_Stroke	M	I	The Unicode radical-stroke count, based or
DerivedCoreProperties.txt			

Lowercase	B	I	Characters with the Lowercase property. Fo see *Chapter 4, Character Properties* in [Unic *Generated from: Ll + Other_Lowercase*
Uppercase	B	I	Characters with the Uppercase property. Fo see *Chapter 4, Character Properties* in [Unic *Generated from: Lu + Other_Uppercase*
Cased	B	I	Characters which are considered to be eithe lowercase or titlecase characters. This prop to the Changes_When_Casemapped propert information, see D120 in *Section 3.13, Defa* in [Unicode]. *Generated from: Lowercase + Uppercase +*
Case_Ignorable	B	I	Characters which are ignored for casing pu information, see D121 in *Section 3.13, Defa* in [Unicode]. *Generated from: Mn + Me + Cf + Lm + Sk Word_Break=MidLetter + Word_Break=Midl*
Changes_When_Lowercased	B	I	Characters whose normalized forms are not toLowercase mapping. For more informatio *Section 3.13, Default Case Algorithms* in [U *Generated from: toLowercase(toNFD(X)) !=*
Changes_When_Uppercased	B	I	Characters whose normalized forms are not toUppercase mapping. For more informatio *Section 3.13, Default Case Algorithms* in [U *Generated from: toUppercase(toNFD(X)) !=*
Changes_When_Titlecased	B	I	Characters whose normalized forms are not toTitlecase mapping. For more information, *3.13, Default Case Algorithms* in [Unicode]. *Generated from: toTitlecase(toNFD(X)) != to*
Changes_When_Casefolded	B	I	Characters whose normalized forms are not folding. For more information, see D127 in *Case Algorithms* in [Unicode]. *Generated from: toCasefold(toNFD(X)) != to*
Changes_When_Casemapped	B	I	Characters which may change when they ur For more information, see D128 in *Section Algorithms* in [Unicode].

			*Generated from: Changes_When_Lowercase Changes_When_Uppercased(X) or Changes_*
Alphabetic	B	I	Characters with the Alphabetic property. Fo see *Chapter 4, Character Properties* in [Unic *Generated from: Lu + Ll + Lt + Lm + Lo + Other_Alphabetic*
Default_Ignorable_Code_Point	B	N	For programmatic determination of default points. New characters that should be ignor (unless explicitly supported) will be assigne permitting programs to correctly handle the such characters when not otherwise suppor information, see the FAQ Display of Unsupp and *Section 5.21, Default Ignorable Code P* *Generated from Other_Default_Ignorable_Code_Point + Cf (format characters) + Variation_Selector - White_Space - FFF9..FFFB (annotation characters) - 0600..0603, 06DD, 070F, 110BD (except that should be visible)*
Grapheme_Base	B	I	For programmatic determination of grapher boundaries. For more information, see Unic #29, "Unicode Text Segmentation" [UAX29]. *Generated from: [0..10FFFF] - Cc - Cf - Cs - Grapheme_Extend*
Grapheme_Extend	B	I	For programmatic determination of grapher boundaries. For more information, see Unic #29, "Unicode Text Segmentation" [UAX29]. *Generated from: Me + Mn + Other_Grapher* **Note:** Depending on an application's interp (private use), they may be either in Grapher Grapheme_Extend, or in neither.
Grapheme_Link (Deprecated as of 5.0.0)	B	I	Formerly proposed for programmatic deter grapheme cluster boundaries. *Generated from: Canonical_Combining_Cla*
Math	B	I	Characters with the Math property. For mor *Chapter 4, Character Properties* in [Unicode

			Generated from: Sm + Other_Math
ID_Start	B	I	Used to determine programming identifiers Unicode Standard Annex #31, "Unicode Ide 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 composi explicitly in CompositionExclusions.txt, plu of *Singleton Decompositions* and *Non-Star* 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 cl specified normalization form.
FC_NFKC_Closure (Deprecated as of 6.0.0)	S	N	Characters that require extra mappings for Folding plus Normalization Form KC. ~~Chara this property have a third field with the ma~~ The mapping is listed in Field 2. ~~*Generated with the following, where **Fold** i default fold operation (excluding the Turki*~~ ~~`b = NFKC(Fold(a));`~~ ~~`c = NFKC(Fold(b));`~~ ~~`if (c != b) add mapping from a to c`~~ ~~`to the set of mappings that constitute the`~~ ~~**Note:** The FC_NFKC_Closure value is omitte is the same as the code point itself.~~
NFD_Quick_Check NFKD_Quick_Check NFC_Quick_Check NFKC_Quick_Check	E	N	For property values, see Decompositions an (Abbreviated names: NFD_QC, NFKD_QC, N
NFKC_Casefold	S	I	~~Mapping from a character to the string proc it, removing any Default_Ignorable_Code_Po and converting to NFKC form. (This set of t repeated, to deal with certain edge cases.)~~ / for best behavior when doing caseless matc interpreted as identifiers. (Abbreviated nam For the definition of the related string trans toNFKC_Casefold() based on this mapping, *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 N mapping. *Generated from: (cp != NFKC_CaseFold(cp))*
PropList.txt			
ASCII_Hex_Digit	B	N	ASCII characters commonly used for the rep hexadecimal numbers.
Bidi_Control	B	N	Format control characters which have speci Unicode Bidirectional Algorithm [UAX9].
Dash	B	I	Punctuation characters explicitly called out Unicode Standard, plus their compatibility e these have the General_Category value Pd, b General_Category value Sm because of their
Deprecated	B	N	For a machine-readable list of deprecated c characters will ever be removed from the st usage of deprecated characters is strongly d
Diacritic	B	I	Characters that linguistically modify the me character to which they apply. Some diacriti characters, and some combining characters
Extender	B	I	Characters whose principal function is to ex shape of a preceding alphabetic character. T length and iteration marks.
Hex_Digit	B	I	Characters commonly used for the represer hexadecimal numbers, plus their compatibi
Hyphen (Stabilized as of 4.0.0; Deprecated as of 6.0.0)	B	I	Dashes which are used to mark connections words, plus the *Katakana middle dot*. The *K* functions like a hyphen, but is shaped like a dash.
Ideographic	B	I	Characters considered to be CJKV (Chinese, and Vietnamese) ideographs. This property class of "Chinese characters" and does not i other logographic scripts such as Cuneiforr Hieroglyphs.
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 speci control of cursive joining and ligation.
Logical_Order_Exception	B	N	~~There are a small number of characters that order. These characters require special hand processing.~~ A small number of spacing vow in certain Southeast Asian scripts such as T use a visual order display model. These lett ahead of syllable-initial consonants, and re handling for processes such as searching ar
Noncharacter_Code_Point	B	N	Code points permanently reserved for inter
Other_Alphabetic	B	C	Used in deriving the Alphabetic property.
Other_Default_Ignorable_Code_Point	B	C	Used in deriving the Default_Ignorable_Cod

Other_Grapheme_Extend	B	C	Used in deriving the Grapheme_Extend pro
Other_ID_Continue	B	C	Used for backward compatibility of ID_Cont
Other_ID_Start	B	C	Used for 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 Uni
Pattern_White_Space	B	N	#31, "Unicode Identifier and Pattern Syntax"
Quotation_Mark	B	I	Punctuation characters that function as quo
Radical	B	N	Used in Ideographic Description Sequences.
Soft_Dotted	B	N	Characters with a "soft dot", like *i* or *j*. An a these characters causes the dot to disappea *above* can be added where required, such a
STerm	B	I	Sentence Terminal. Used in Unicode Standa "Unicode Text Segmentation" [UAX29].
Terminal_Punctuation	B	I	Punctuation characters that generally mark units.
Unified_Ideograph	B	N	A property which specifies the exact set of Ideographs in the standard. This set exclud Ideographs (which have canonical decompo Ideographs), as well as characters from the Punctuation block. The property is used in Ideographic Description Sequences.
Variation_Selector	B	N	Indicates characters that are Variation Selec the behavior of these characters, see Standa *Section 16.4, Variation Selectors* in [Unicod Standard Annex #37, "Unicode Ideographic [UTS37].
White_Space	B	N	Spaces, separator characters and other con should be treated by programming languag for the purpose of parsing elements. See al Grapheme_Cluster_Break, Sentence_Break, a which classify space characters and related differently for particular text segmentation ~~**Note:** ZERO WIDTH SPACE and ZERO WIDTH are not included, because their functions ar line-break control. Their names are unfortu this respect.~~ ~~**Note:** There are other senses of "whitespac different set of characters.~~
UnicodeData.txt			
Name	M	N	(1) These names match exactly the names p charts of the Unicode Standard. The derivec names are omitted from this file; see Jamo.t derivation.

General_Category	E	N	(2) This is a useful breakdown into various which can be used as a default categorizati implementations. For the property values, s Values.
Canonical_Combining_Class	N	N	(3) The classes used for the Canonical Orde Unicode Standard. This property could be c enumerated property or a numeric property the property is in terms of the numeric valu value names associated with different nume DerivedCombiningClass.txt and Canonical C Values.
Bidi_Class	E	N	(4) These are the categories required by the Bidirectional Algorithm. For the property va Bidirectional Class Values. For more inform Standard Annex #9, "The Unicode Bidirectio [UAX9]. The default property values depend on the explained in DerivedBidiClass.txt
Decomposition_Type Decomposition_Mapping	E S	N	(5) This field contains both values, with the brackets. The decomposition mappings exa decomposition mappings published with th the Unicode Standard. For more informatio Decomposition Mappings.
Numeric_Type Numeric_Value	E N	N	(6) If the character has the property value Numeric_Type=Decimal, then the Numeric_ represented with an integer value in fields 6 discussion of *decimal digits* in *Chapter 4, C* in [Unicode].
	E N	N	(7) If the character has the property value N then the Numeric_Value of that digit is repr integer value in fields 7 and 8, and field 6 i digits that need special handling, such as th superscript digits.
	E N	N	(8) If the character has the property value Numeric_Type=Numeric, then the Numeric_ character is represented with a positive or r rational number in this field, and fields 6 ar includes fractions such as, for example, "1 / VULGAR FRACTION ONE FIFTH. Some characters have these properties base Unihan data files. See Numeric_Type, Han.
Bidi_Mirrored	B	N	(9) If the character is a "mirrored" character this field has the value "Y"; otherwise "N". S *Mirrored—Normative* of [Unicode]. *Do not c Bidi_Mirroring_Glyph property.*

Unicode_1_Name	M	I	(10) Old name as published in Unicode 1.0. provided when it is significantly different fr for the character. The value of field 10 for c does not always match the Unicode 1.0 nan contains ISO 6429 names for control functi the code charts.
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 appeared in parentheses in the 10646 name an asterisk to mark an Annex P note. As of Unicode 5.2.0, this field no longer co values.
Simple_Uppercase_Mapping	S	N	(12) Simple uppercase mapping (single char If a character is part of an alphabet with cas has a simple uppercase equivalent, then the equivalent is in this field. The simple mappi character result, where the full mappings m character results. For more information, see Mapping.
Simple_Lowercase_Mapping	S	N	(13) Simple lowercase mapping (single char
Simple_Titlecase_Mapping	S	N	(14) Simple titlecase mapping (single charac **Note:** If this field is null, then the Simple_T the same as the Simple_Uppercase_Mapping

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 purely as a reformatting of data for properties specified in other data files. In case of any inadvertant mismatch between the primary data files specifying those properties and these lists of extracted properties, the primary data files are taken as definitive.

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 2
DerivedJoiningType.txt	N	Joining_Type	ArabicShaping.txt, field 1
DerivedLineBreak.txt	N	Line_Break	LineBreak.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.

Numeric_Value is extracted based on the actual numeric value of the data in field 8 of UnicodeData.txt or 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~~3.1~~ 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 in the property table.

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—Normative in [Unicode] introduces case and case mapping properties. *Table 4-1, 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 ~~the~~ *Table 9, Property Table* ~~above~~.

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 digraphic character, with first part uppercase
Lm	Modifier_Letter	a modifier letter
Lo	Other_Letter	other letters, including syllables and ideographs
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
Nd	Decimal_Number	a decimal digit
Nl	Letter_Number	a letterlike numeric character
No	Other_Number	a numeric character of other type
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
Sm	Math_Symbol	a symbol of primarily mathematical use
Sc	Currency_Symbol	a currency sign

Sk	Modifier_Symbol	a non-letterlike modifier symbol
So	Other_Symbol	a symbol of other type
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
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

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

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.

The symbol "L&" is 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. The LC value for the General_Category property, as documented in PropertyValueAliases.txt also stands for uppercase, lowercase or titlecase letters.

The Unicode Standard does not assign non-default 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
L	Left_To_Right	any strong left-to-right character
LRE	Left_To_Right_Embedding	U+202A: the LR embedding control
LRO	Left_To_Right_Override	U+202D: the LR override control
R	Right_To_Left	any strong right-to-left (non-Arabic-type) character
AL	Arabic_Letter	any strong right-to-left (Arabic-type) character
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
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
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

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

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 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.

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 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>.

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 all the long symbolic aliases for Canonical_Combining_Class values, reproduced from PropertyValueAliases.txt, along with a brief description of each category.

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
7	Nukta	Diacritic nukta marks in Brahmi-derived scripts
8	Kana_Voicing	Hiragana/Katakana voicing marks
9	Virama	Viramas
10		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 top 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, but these two sets are not congruent. 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.

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, ~~"Canonical Ordering Behavior"~~ Normalization Forms* in [Unicode].

5.7.5 Decompositions and Normalization

Decomposition is specified in *Chapter 3, Conformance* of [Unicode]. ~~UAX #15, Unicode Normalization Forms [UAX15]~~ That chapter also specifies the interaction between decomposition and normalization. ~~That annex specifies how the decompositions defined in UnicodeData.txt are used to derive normalized forms of Unicode text.~~

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.

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.

5.8.1 Property Aliases

In PropertyAliases.txt, the first field 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 short and less mnemonic, but is useful for expressions such as "lb=BA" in data or in other contexts where the meaning is clear.

The property aliases specified in PropertyAliases.txt constitute a unique name space. 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 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 name space. 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 value, 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.

The property value alias entries for three properties, Age, Block, and Joining_Group, make use of a special metavalue "n/a" in the field for the abbreviated alias. This should be understood as meaning that no abbreviated alias is defined for that value for that property, rather than as an alias per se.

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.

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.

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".

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 letters in the normative Unicode character name, as published in the Unicode names list, 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. Thus the hyphen in the name U+10089 LINEAR B IDEOGRAM B107M HE-GOAT is medial, and should be ignored in loose matching, but the hyphen in the name U+0F39 TIBETAN MARK TSA -PHRU is *not* medial, and should not be ignored in loose matching.

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 formal name aliases and the names of named character sequences, which share the namespace (and matching behavior) of Unicode character names.

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 ('_'), ~~and~~ 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 properties, which do not have symbolic aliases defined for their values; exact matching for String 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. Also, some multi-valued properties such as Line_Break or Word_Break may have additional values defined for them. 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
Name	na
Jamo_Short_Name	jsn
Canonical_Combining_Class	ccc
Decomposition_Mapping	dm
Pattern_Syntax	Pat_Syn
Pattern_White_Space	Pat_WS

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.

All characters other than those of General_Category M* are guaranteed to have Canonical_Combining_Class=0. Currently it is also true that all characters other than those of General_Category Mn have Canonical_Combining_Class=0. However, the more constrained statement is not a guaranteed invariant; it is possible that some new character of General_Category Me or Mc could be given a non-zero value for Canonical_Combining_Class in the future.

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..255. 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 5.1 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 256 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 ints 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 Combining_Character_Class Property

The Combining_Character_Class (ccc) property is a hybrid type. The possible values defined for it in UnicodeData.txt range from 0 to 255 and are numeric values. However, Combining_Character_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 Combining_Character_Class, concatenate together the symbolic aliases from PropertyValueAliases.txt, and then add the numeric range 0..255.

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 Age, 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
$positiveDecimal	[0-9]+\.[0-9]+
$decimal	-?$positiveDecimal
$optionalDecimal	-?[0-9]+(\.[0-9]+)?
$name	[a-zA-Z0-9]+([_-\ ][a-zA-Z0-9]+)*
$codePoint	(10|[A-F0-9])?[A-F0-9]{4}

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
blk	Block	/$name/	
sc	Script		
dm	Decomposition_Mapping	/$codePoint+/	
FC_NFKC	FC_NFKC_Closure		
NFKC_CF	NFKC_Casefold		
cf	Case_Folding	/$codePoint+/	
lc	Lowercase_Mapping		
tc	Titlecase_Mapping		
uc	Uppercase_Mapping		
sfc	Simple_Case_Folding	/$codePoint/	
slc	Simple_Lowercase_Mapping		
stc	Simple_Titlecase_Mapping		
suc	Simple_Uppercase_Mapping		
bmg	Bidi_Mirroring_Glyph	/$codePoint/	

na	Name	/$name/

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 ~~the~~ *Table 9, Property Table*. For example, see the Grapheme_Link property.

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
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].)

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].) 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. 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 BidiTest.txt

This file contains data which can be used to test an implementation of the Unicode Bidirectional Algorithm. (See [UAX9].)

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 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 should not deviate from those results. Tailored implementations can also use the data in BidiTest.txt 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. ~~The summary includes changes extending all the way back to Unicode 2.0.0, taken from the obsoleted UCD.html documentation file, which predates the creation of this annex. The intent is for this first consolidated version of the annex to preserve that complete prior history from UCD.html. Subsequent versions of the annex will provide only an abbreviated UCD change history section containing only the delta change information from each preceding version.~~

~~Starting from Unicode 4.0.1,~~ References in the change history are often made to a Public Review Issue (PRI). See http://www.unicode.org/review/resolved-pri.html for more information about each of those cases.

~~Changes documented prior to Unicode 4.0 only covered UnicodeData.txt. From Unicode 4.0 onward, the documentation of changes includes modifications of other files as well.~~

Unicode 6.0.0

General:

[TBD]

Changes in specific files:

[TBD]

Unicode 5.2.0

General:

The documentation file UCD.html was obsoleted. The main documentation for the UCD is now contained in [UAX44]. Documentation specifically for the Unihan data files can be found in [UAX38]

Changes in specific files:

Appropriate data files were updated to include the 6,648 new characters added in Unicode 5.2. Nine new properties were added.

- UnicodeData.txt
  - U+2071 and U+207F were changed from gc=Ll to gc=Lm.
  - The Bidi_Class value for mathematical partial differential symbols was changed from bc=L to bc=ON: U+1D6DB, U+1D715, U+1D74F, U+1D789, and U+1D7C3.
  - Case mappings were added for U+023F, U+0240, and U+0252, to map to newly-encoded uppercase letters.
  - The ISO_Comment property was obsoleted, and all values changed to the null string.
- ArabicShaping.txt
  - Two new joining groups, FARSI YEH and NYA, were added.
- LineBreak.txt
  - U+1B5C was changed from lb=BA to lb=AL.
  - U+09F2 and U+09F3 were changed from lb=PR to lb=PO.
  - U+09F9 was changed from lb=AL to lb=PO.
  - Added a new Line_Break property value CP, and changed U+0029 and U+005D to use that value.
- PropList.txt
  - After a review of the meaning of the Deprecated property, several characters had their values changed. Correlated changes were made to annotations in the code charts.
  - Changed to Deprecated=True: 0149, 0F77, 0F79, 2329, 232A.
  - Changed to Deprecated=False: 0340, 0341, 17D3.
- PropertyAliases.txt
  - Added aliases for new properties: Cased, Case_Ignorable, Changes_When_Casefolded, Changes_When_Casemapped, Changes_When_NFKC_Casefolded, Changes_When_Lowercased, Changed_When_Titlecased, Changes_When_Uppercased, and NFKC_Casefold.
  - Added an alias for the property Name_Alias.
  - Added aliases for all normative and informative properties from the Unihan data files, based on the values for the Unihan tags.

    - Updated the aliases for the Unicode_Radical_Stroke property, to match the pattern for other Unihan tags.
- PropertyValueAliases.txt
    - Added aliases for the values of all the new character properties.
    - Added @missing statements for the default values for all of the normative and informative properties from the Unihan data files.
- Scripts.txt
    - Characters with the Script property value Inherited were updated to make use of the new alias Zinh (changed from Qaai).
- DerivedCoreProperties.txt
    - Added new derived property listings for Cased, Case_Ignorable, Changes_When_Lowercased, Changes_When_Uppercased, Changes_When_Titlecased, Changes_When_Casefolded, and Changes_When_Casemapped.
- DerivedNormalizationProps.txt
    - Added new derived property listings for NFKC_Casefold and Changes_When_NFKC_Casefolded.
- NamedSequences.txt
    - Tamil named sequences were updated from provisional to approved, and moved into this file.
- NamedSequencesProv.txt
    - Added a provisional named sequence for BENGALI LETTER KHINYA.
- SentenceBreakProperty.txt
    - The Sentence_Break property values of two superscript letters were changed from wb=Lower to sb=OLetter to harmonize better with other letter modifiers. This affected U+2071 and U+207F.
- WordBreakProperty.txt
    - Changed the value for U+200B ZERO WIDTH SPACE from wb=Format to wb=Other.
- Text Boundary Test Files
    - New test cases were added to LineBreakTest.txt to account for the addition of the new Line_Break property value lb=CP and its implications for line breaking.
- BidiTest.txt
    - This new test file was added, with test cases for UAX #9.
- CJKRadicals.txt
    - This new data file was added.

Acknowledgments

Mark Davis and Ken Whistler are the authors of the initial version and have added to and maintained the text of this annex. 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 5

- Proposed update for Unicode 6.0.0.
- Removed old UCD Change History entries prior to Unicode 5.2.0.
- Updated status of Hyphen and ISO_Comment properties to Deprecated.
- Updated status of several derived normalization properties to Deprecated.
- Added tables listing Deprecated and Stabilized properties.
- Extended the discussion of the significance of the Bidi_Mirroring_Glyph property.
- Clarified the intended application of the Ideographic and Unified_Ideograph properties.
- Moved Property Summary to top of Section 5, renamed it to Property Index, and adjusted Section 5 numbering.
- Renumbered tables to account for two table insertions.
- Rewrote the description of the Logical_Order_Exception and White_Space properties for clarity.
- Added clarification for UAX44-LM2 in Matching Rules.
- Updated matching rule UAX44-LM3 to ignore initial "is" in Matching Rules.
- Added U+110BD to the list of exceptions to the derivation of Default_Ignorable_Code_Point.
- Added anchors to the matching rules.
- Updated the description fields for FC_NFKC_Closure and NFKC_Casefold.
- Miscellaneous minor point edits.

Revision 4 [KW]

- **Reissued** for Unicode 5.2.0.
- Completely reorganized and rewritten, to include all the content from the obsoleted UCD.html.
- Added Section 5.10 re deprecation.
- Added subsection in Section 4.2 re line termination conventions.
- Added Contributory as a formal status and updated the Property Table accordingly.
- Added note in Section 5.3.1 to indicate that contributory properties are neither normative nor informative.
- Updated documentation for default values.
- Cleaned up description of numeric properties.
- Tweaked the description of NamesList.html.
- Miscellaneous minor point edits.
- Updated summary statement of the document.

- Centered tables.
- Added anchors and numbers to tables and adjusted text referencing tables accordingly.
- Added clarifications about exceptional format issues for Unihan data files.
- Updated references to *Section 4.8, Named—Normative* for derived names and for code point labels.
- Added mention of property aliases from Unihan data files to Section 5.6.1.
- Added documentation for new derived properties: Cased, Case_Ignorable, Changes_When_Lowercased, Changes_When_Uppercased, Changes_When_Titlecased, Changes_When_Casefolded, Changes_When_Casemapped, NFKC_Casefold, and Changes_When_NFKC_Casefolded.
- Added strong pointers to Section 3.5 and Chapter 4 of [Unicode] in the Introduction.
- Added new *Section 2.3.1, Changes to Properties Between Releases*.
- Updated default values for East_Asian_Width.
- Clarified the applicability of comments in cases where properties have multiple default values.
- Restructured Section 5.1 documentation of columns in the property table, for better text flow.
- Reordered entries for DerivedCoreProperties.txt in the property table, for clarity.
- Added documentation of new test file: BidiTest.txt.
- Updated terminology related to the Unihan Database.
- Added documentation for the new data file, CJKRadicals.txt.
- Added Attached_Above for ccc=214 in Table 13.
- Complete revision of Validation section and associated tables.
- Minor revision of text in *Section 4.1.5, File Directory Differences for Early Releases*.
- Added a cautionary note about the use of the Age property in regular expressions.
- Added sections explaining obsolete, deprecated, and stabilized properties, and clearly identified existing such properties in the property table.

Revision 3 being a proposed update, only changes between versions 4 and 2 are noted here.

Revision 2

- Initial approved version for Unicode 5.1.0.

Revision 1

- Initial draft.
