

Proposed Draft Unicode Technical Report #23

CHARACTER PROPERTIES

Version	1.0
Author	Asmus Freytag (asmus@unicode.org)
Date	2002-04-23
This Version	http://www.unicode.org/unicode/reports/tr23/tr23-1.html
Previous Version	none
Latest Version	http://www.unicode.org/unicode/reports/tr23/
Tracking Number	<u>1</u>

Summary

This report presents a survey of the character properties defined in the Unicode Standard as well as guidelines to their usage.

Status

*This document has been approved by the Unicode Technical Committee for public review as a **Proposed Draft Unicode Technical Report**. Publication does not imply endorsement by the Unicode Consortium. This is a draft document which may be updated, replaced, or superseded by other documents at any time. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.*

Significant changes of the information in this proposed draft are expected as result of public and UTC review.

Please send comments to the authors. A list of current Unicode Technical Reports is found on <http://www.unicode.org/unicode/reports/>. For more information about versions of the Unicode Standard, see <http://www.unicode.org/unicode/standard/versions/>.

The [References](#) provide related information that is useful in understanding this document. Please mail corrigenda and other comments to the author(s).

[Editorial notes for the benefit of reviewers are indicated like this.]

Contents

[1 Scope](#)

[2 Overview](#)

[2.1 Origin of Character Properties](#)

[2.2 Character Behavior in Context](#)

[2.3 Relation of Character Properties to Algorithms](#)

[2.4 Normative Properties](#)

[2.5 Informative Properties](#)

[2.6 Issues](#)

[3 Definitions](#)

[4 Conformance](#)

[4.1 Conformance Requirements](#)

[4.2 Overriding Properties and Higher-level Protocols](#)

[4.3 Normative and Informative Properties](#)

[7 Updating Character Properties and Extending the Standard](#)

[7.1 Updating](#)

[7.2 Guarantees](#)

[8 Special Property Values](#)

[8.1 N/A](#)

[8.2 Default Values](#)

[8.3 Preliminary Property Assignments](#)

[9 Working with properties](#)

[9.1 Subsection 2.1](#)

[9.2 Subsection 2.2](#)

[9.3 Subsection 2.1](#)

[9.4 Subsection 2.2](#)

[10 Data Management and Distribution](#)

[10.1 Versions](#)

[10.2 File Syntax Conventions](#)

[10.3 Beta](#)

[11. References](#)

[Acknowledgements](#)

[Revisions](#)

1. Scope

This survey provides discussion of common aspects of character properties. This survey is not intended to supersede chapter 4 in the book, nor the existing body of technical reports and documentation files in the Unicode Character Database that provide detailed descriptions for particular character properties. Instead it presents a capsule summary only and references the corresponding technical report for the full details.

This report specifically covers formal **character properties**, which are those attributes of characters that are specified according to the definitions set forth in this report.

2. Overview

2.1 Origin of Character Properties

The Unicode Standard views character semantics as inherent to the definition of a character and conformant processes are required to take these into account when interpreting characters. The assignment of character semantics for the Unicode Standard is based on character behavior. For other character set standards, it is left to the implementer, or to unrelated secondary standards, to assign character semantics to characters. In contrast, the Unicode Standard supplies a rich set of character attributes, called properties, for each character contained in it. Many properties are specified in relation to processes or algorithms that interpret them, in order to implement the discovered character behavior.

2.2 Character Behavior in Context

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

Other examples that require context include the classification of neutrals in script assignments or title casing. The line breaking rules of TR#14 involve character pairs and and triples, and in certain cases, longer sequences. The glyph(s) defined by a combining character sequence are the result of contextual analysis in the display shaping engine. Isolated character properties typically only tell part of the story.

2.3 Relation of Character Properties to Algorithms

When modeling character behavior with computer processes, formal character properties are assigned in order to achieve the expected results. Such modeling depends heavily on algorithms. In some cases, a given character property is specified in close conjunction with a detailed specification of an algorithm. In other cases, algorithms are implied but not specified, or there are several algorithms can make use of the same general character property. The last case may require occasional differences in character property assignment to make all algorithms work correctly. This can usually be achieved by overriding specific properties for specific algorithms.

When assigning character properties for use with a given algorithm, it may be tempting to assign somewhat values to some characters, as long as the algorithm happens to produce the expected results. Proceeding in this way hides the nature of the character and limits the re-use of character properties by related processes. Therefore, instead of tweaking the properties to simply make a particular algorithm easier, the Unicode Standard pays careful attention to the underlying essential linguistic identity of the character. However, not all aspects of a characters identity are relevant in all circumstances, and some characters can be used in many different ways, depending on context or circumstance. Because of this the formal character properties alone are not sufficient to describe the complete range of desirable or acceptable character behaviors.

2.4 Normative Properties

As specified in Chapter 3, Conformance, the Unicode Standard [UNICODE] defines both normative and informative properties.

Normative Properties. *Normative* means that implementations that claim conformance to the Unicode Standard (at a particular version) and that make use of a particular property must follow the specifications of the standard for that property to be conformant. The term *normative* when applied to a character property does *not* mean that the value of the property will never change. Corrections and extensions to the standard in the future may require minor changes to normative values, even though the Unicode Technical Committee strives to minimize such changes.

By making a property normative, the Unicode Standard guarantees that conformant implementations can rely on the fact that other conformant will interpret the character in the same way. This is most useful for those properties where the Unicode Standard provides precise rules for the interpretation of characters based on their properties. An example are the bidirectional properties and their use by the bidirectional algorithm. For some character properties, for example the general category, the Unicode standard does not define what model of processing it is intended to support and what the required consequences are of a character being e.g. "Letter Other" as opposed to "Symbol Other". In the absence of such definition, the only effect of conformance that can be tested in a strict manner is whether a character property library returns the correct value to its caller.

Note: one trivial, but important instance of conformant implementation is runtime access to a character property database. For normative properties, conformant implementations guarantee that the returned values match the values defined by the Unicode Consortium.

2.5 Informative Properties

Informative Properties. An *informative* character property is strongly recommended, but a conformant implementation is free to use or change such values as it may require, while still remaining conformant to the standard. Particular implementations may choose to

override the properties that are not normative. In that case, the implementer has the option of establishing a protocol to convey that information.

Properties may be informative for two main reasons.

1. The nature of the property or the precise set of characters to which it applies are not yet definite and it therefore is too early to assign a normative property. Even if there was a precise description of how to interpret such a property, the fact that it is subject to a (planned) revision makes it less interesting to communicating implementations to rely on the specified behavior.
2. Existing implementations show a range of behaviors for the same character, many or all of which may be equally useful choice on the part of their designers. Assigning a normative property would imply a unwarranted restriction on existing and established practice.

3. Definitions

PD1. Code Point Property

A code point property defines a set of values and a mapping from each Unicode code point to one of the values of the set.

PD2. Character Property

A character property defines a set of values and a mapping from each Unicode character to one of the values of the set.

Character Properties typically map a default value to any code point not assigned to a character.

PD3. Property Value

One of the set of values associated with a character property.

For example, the [\[East Asian Width\]](#) property has the possible values "Narrow", "Neutral", "Wide", "Ambiguous" and "Unassigned".

PD4. Universal (Required) Property

A universal property applies to all Unicode characters. A universal property does not have a 'does not apply value'.

PD5. Limited Property

A property which is not defined or does not apply to all characters is called a *limited* property. It applies to only a subset of Unicode characters, usually one script, or a related family of scripts.

For example case, and case mapping information is not needed for unicameral scripts. Such information can in principle be left 'undefined' for the characters to which it does not apply. Limited properties are often implemented by listing property values for all characters and giving a special 'does-not-apply' value to all characters to which the limited property does not apply.

PD6. Enumerated Property

A property with a fixed set of values. This is sometimes also known as a partition.

As characters are added to the Unicode Standard, the set of values may need to be extended in the future, but it is advantageous to think of enumerated properties of having a fixed set of possible values.

PD7. Closed Enumeration

An enumerated property for which the set of values is closed (i.e. it may not be extended for future versions of the Unicode Standard).

Note: Currently, the General Category is the only closed enumeration, other than Boolean properties.

PD8. Single Valued (Boolean) Property

A closed enumerated property whose set of values is limited to 'true' and false.

Essentially the presence or absence of the property is the important information.

PD9. Integral Property

An integral property can take on any integer or real value. An example is the **decimal digit value** property. There is no implied limit to the number of possible distinct values for the property, short of the limitations of representing integers in computers.

PD10. General Numeric Property

A general numeric property can take on any integer, real, or complex value. An example is the **numeric value** property. There is no implied limit to the number of possible distinct values for the property, short of the limitations of representing integers, real or complex numbers in computers.

PD11. Normative Property

A normative property has conformance implications, see 4. Conformance. A normative character property must be paired with a precise description of how to interpret characters with each normative property value in a conformant way.

Note: A normative process that depends in a normative and testable way on a property, causes the property to be normative. For example, the interpretation of the [bidirectional class](#) is precisely defined in [[Bidirectional Algorithm](#)].

If a process does not interpret a given character, it may remain unaware of its properties – but it is recommended that processes use carefully chosen default values for characters that they don't handle.

PD12. Informative Property

An informative property is provided as helpful information to implementers. There are no requirements to implementations of the Unicode Standard.

Note: Informative properties capture expert implementation experience and their use is strongly recommended by the Consortium.

PD13. Simple Property

A property that applies to a character in isolation.

PD14. Character Behavior

A property that applies to a character in context of a longer character sequence

PD15. Stable Property

A property is stable with respect to a particular algorithm or process, if changes in the assignment of property values produce no changes in the outcome of the process or algorithm.

For example, while the absolute values of the canonical combining classes are not

guaranteed to be the same between versions of the Unicode Standard, their relative values will be maintained. As a result, they are stable with respect to the Normalization Forms as defined in [Normalization].

PD16. Immutable Property

A property whose values, once assigned to a character, are fixed and will not be changed.

An example of immutable, or fixed, properties are the code position and name of each Unicode character.

PD17. Overridable Property

A property whose values may be overridden by a higher level protocols.

PD18. Default Value

Value of a property to be used when encountering unassigned or unsupported characters.

There may be more than one default value per property.

4. Conformance related considerations

4.1 Conformance Requirements

In Chapter 3, Conformance, The Unicode Standard [Unicode] states that *"A process shall interpret a coded character representation according to the character semantics established by this standard, if that process does interpret that coded character representation."* The semantics of a character are established by taking its coded representation, character name and representative glyph in context and are further defined by its normative properties and behavior. Neither character name nor representative glyphs can be relied upon absolutely; a character may have a broader range of use than the most literal interpretation of its character name, and the representative glyph is only indicative of one of a range of typical glyphs representing the same character.

4.2 Overriding properties via Higher-level Protocols

The Unicode Standard makes these specific statements about overriding properties:

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

- The character combination properties and the canonical ordering behavior cannot be overridden by higher-level protocols.
- Particular implementations may choose to override all properties that are not normative.

For interpreting directionality, higher-level protocols may:

- Override the number handling to use information provided by a broader context. For example, information from other paragraphs in a document could be used to conclude that the document was fundamentally Arabic and that EN should generally be converted to AN.
- Replace, supplement, or override the directional overrides or embedding codes. This task is accomplished by providing information via additional stylesheet or markup information about the embedding level or character direction. The interpretation of such information must always be defined by reference to the behavior of the equivalent explicit codes as given in the algorithm.

- Override the bidirectional character types assigned to control codes to match the interpretation of the control codes within the protocol. (See also Section 13.1, Control Codes.)
- Remap the number shapes to match those of another set. For example, remap the Arabic number shapes to have the same appearance as the European numbers.

[Ed. Note: Section 5 and 6 have been removed from this draft]

7. Updating Properties and Extending the Standard

7.1 Updating Properties

Updates to the Unicode Character Database can be required for three reasons

1. To cover new characters added to the Unicode Standard
2. To add new properties
3. To change the assigned values for a property for some characters

Changing a characters property assignment invalidates existing implementations and is therefore something that is done judiciously and with great care when there is no better alternative.

- **Unless explicitly stated otherwise, The Unicode Consortium may change the properties of characters.**

The consortium will endeavor to keep the values of all character properties as stable as possible, but some circumstances may arise that require changing them. In particular, as Unicode encodes less-well documented scripts (such as for minority languages in Thailand) the exact character properties and behavior may not be known at the time the script is first encoded.

7.2 Guarantees

For some properties, some of the following aspects are guaranteed to be invariant.

- stability of assignment
- stability of set of values for a property
- stability of file formats
- stability of relation to another property

The status of a property as normative does not imply a stability guarantee.

- **Once a character is encoded, its code position and name are immutable properties.**

Mistakes in naming are noted in the nameslist in a note or by using an alias, but the formal name remains unchanged.

- **Once a character is encoded, its canonical combining class and decomposition (canonical or combining) are stable with respect to normalization.**

If a string contains only characters from a given version of the Unicode Standard (say Unicode 3.1), and it is put into a normalized form in accordance with that version of Unicode, then it will be in normalized form when according to any past or future versions of Unicode.

***Note:** If an implementation normalizes a string that contains characters that are **not** assigned in the version of Unicode that it supports, that string may not be assessed as being*

in normalized form according to a future version of Unicode. For example, suppose that a Unicode 3.0 program normalizes a string that contains new Unicode 3.1 characters. That string may not be normalized according to Unicode 3.1.

- **The General Category and Bidi Category are closed enumerations.**

In other words they will not be further subdivided.

- **The structure of certain property values in the Unicode Character Database will not be changed:**

Further description of these is provided in described in [UnicodeData.html](#):

- Combining classes are limited to the values 0 to 255.
- All characters other than those of General Category M* have the combining class 0.
- Canonical and Compatibility mappings are always in canonical order, and the resulting recursive decomposition will also be in canonical order.
- Canonical mappings are always limited either to a single value or to a pair. The second character in the pair cannot itself have a canonical mapping.

8. Special Property Values

8.1 N/A Value

Limited properties apply to only a subset of characters. Where these properties are implemented as a partition (required property) the characters to which the property does not apply is given a special value denoting that the property does not apply.

8.2 Default Value

Implementations often need specific properties for *all* code points, including those that are unassigned. To meet this need, the Unicode standard assigns default properties to ranges of unassigned code points.

All implementations of the Unicode Standard should endeavor to handle additions to the character repertoire gracefully. In some cases this may require that an implementation attempts to 'anticipate' likely property values for Code points for which characters have not yet been defined, but where surrounding characters exist that make it probable that similar characters will be assigned to the Code point in question.

There are three strategies

1. Rely on the recommendation from The Unicode Consortium. For example, for the Bidirectional Class, the Unicode Consortium has published recommended default values for all code points.
2. Treat the unassigned areas of a block as if they had property values common to other characters of the block. A variation of this scheme bridges 'holes' in the allocation by using the property values for the characters bracketing the hole.
3. Give unassigned code location a different default property that will result in graceful, if not completely correct behavior if encoded characters are later encountered at that location.

Each of these strategies has advantages and drawbacks, and none can guarantee that the behavior of an implementation that is conformant to a prior version of the Unicode Standard will support characters added in a later version of the Unicode Standard in precisely the same way as an implementation that is conformant to the later version. The most that can be hoped for, is that

the earlier implementation will behave gracefully in such circumstances.

Default values are temporary: they will be superseded by final assignments, once characters are assigned to a given code point.

For non-character codes, a property returning API would return the same value as the default value for unassigned characters.

8.3 Undetermined Property Values

For many archaic scripts (as well as for not yet fully implemented modern ones) essential characteristics of many characters may not be knowable at the time of their publication. In these cases the proper assignments of property values for newly encoded characters cannot be reliably determined at the time the characters are first added to the Unicode Standard, or for a new property, when the property is first added to the Unicode Character Database. In these cases, and where the property is a required property, it will be given a value of 'undetermined', or 'unknown at time of publication'.

8.4 Preliminary Property Assignments

Sometimes, a determination and assignment of property values can be made, but the information on which it was based may be incomplete or preliminary. In such cases, the property value may be changed when better information becomes available. Currently, there is no machine readable way to provide information about the confidence of a property assignment; however, the text of the Standard or a Technical Report defining the property may provide general indications of preliminary status of property assignments where they are known.

9. Working with properties

[The text in this section is very preliminary].

There are two main issues in working with properties

- Efficient storage and access to the property information
- Interpretation of the character property and implementation of character behavior in context

9.1 Efficient storage and access to property information

The Unicode Standard provides detailed information on character properties (see *Chapter 4, Character Properties*, and the Unicode Character Database on the accompanying CD-ROM).

These properties can be used by implementers to implement a variety of low-level processes. Fully language-aware and higher-level processes will need additional information.

A two-stage table, as described in *Section 5.1, Transcoding to Other Standards*, can also be used to handle mapping to character properties or other information indexed by character code. For example, the data from the Unicode Character Database on the accompanying CD-ROM can be represented in memory very efficiently as a set of two-stage tables.

Individual properties are common to large sets of characters and therefore lend themselves to implementations using the shared blocks.

Many popular implementations are influenced by the POSIX model, which provides functions for separate properties, such as `isalpha`, `isdigit`, and so on. Implementers of Unicode-based systems and internationalization libraries need to take care to extend these concepts to the full set of Unicode characters correctly.

In Unicode–encoded text, combining characters participate fully. In addition to providing callers with information about which characters have the combining property, implementers and writers of language standards need to provide for the fact that combining characters assume the property of the preceding base character (see also Section 3.5, Combination, and Section 5.16, Identifiers). Other important properties, such as sort weights, may also depend on a character's context.

Because the Unicode Standard provides such a rich set of properties, implementers will find it useful to allow access to several properties at a time, possibly returning a string of bit–fields, one bit–field per character in the input string.

In the past, many existing standards, such as the C language standard, assumed very minimalist "portable character sets" and geared their functions to operations on such sets. As the Unicode encoding itself is increasingly becoming the portable character set, implementers are advised to distinguish between historical limitations and true requirements when implementing specifications for particular text processes.

Multistage Tables

Tables require space. Even small character sets often map to characters from several different blocks in the Unicode Standard, and thus may contain up to 64K entries in at least one direction. Several techniques exist to reduce the memory space requirements for mapping tables. Such techniques apply not only to transcoding tables, but also to many other tables needed to implement the Unicode Standard, including character property data, collation tables, and glyph selection tables.

Flat Tables. If disk space is not at issue, virtual memory architectures yield acceptable working set sizes even for flat tables because frequency of usage among characters differs widely and even small character sets contain many infrequently used characters. In addition, data intended to be mapped into a given character set generally does not contain characters from all blocks of the Unicode Standard (usually, only a few blocks at a time need to be transcoded to a given character set). This situation leaves large sections of the 64K–sized reverse mapping tables (containing the default character, or unmappable character entry) unused—and therefore paged to disk.

Ranges. It may be tempting to "optimize" these tables for space by providing elaborate provisions for nested ranges or similar devices. This practice leads to unnecessary performance penalties on modern, highly pipelined processor architectures because of branch penalties.

A faster solution is to use an optimized two–stage table, which can be coded without any test or branch instructions. Hash tables can also be used for space optimization, although they are not as fast as multistage tables.

Two–Stage Tables (Single index tries). Two–stage (high–byte) tables are a commonly employed mechanism to reduce table size (see Figure 5–1). They use an array of 256 pointers and a default value. If a pointer is NULL, the returned value is the default. Otherwise, the pointer references a block of 256 values. If full support for supplementary characters is required, three–stage tables (or double index tries) are a better solution.

Figure 5–1. Two–Stage Tables [TBD]

Optimized Two–Stage Table. Wherever any blocks are identical, the pointers just point to the same block. For transcoding tables, this case occurs generally for a block containing only mappings to the "default" or "unmappable" character. Instead of using NULL pointers and a default value, one "shared" block of 256 default entries is created. This block is pointed to by all first–stage table entries, for which no character value can be mapped. By avoiding tests and

branches, this strategy provides access time that approaches the simple array access, but at a great savings in storage.

Given an arbitrary 64K table, it is a simple matter to write a small utility that can calculate the optimal number of stages and their width.

Range tables

[additional guidelines, TBD]

11. References

[ArabicShaping]	Data file ftp://ftp.unicode.org/Public/UNIDATA/ArabicShaping.txt
[Bidirectional Algorithm]	Mark Davis, Unicode Standard Annex #9: The Bidirectional Algorithm, http://www.unicode.org/unicode/reports/tr9
[CaseFolding]	Data file ftp://ftp.unicode.org/Public/UNIDATA/CaseFolding.txt
[Case Mapping]	Mark Davis, Unicode Technical Report #21: Case Mapping, http://www.unicode.org/unicode/reports/tr21
[Character Mapping Tables]	Mark Davis, Unicode Technical Report #22: Character Mapping Tables, http://www.unicode.org/unicode/reports/tr22
[Collation]	Mark Davis, Unicode Technical Report #10: Collation, http://www.unicode.org/unicode/reports/tr10
[EastAsianWidth]	Data file ftp://ftp.unicode.org/Public/UNIDATA/EastAsianWidth.txt
[East Asian Width]	Asmus Freytag, <i>Unicode Standard Annex #11, East Asian Width</i> , http://www.unicode.org/unicode/reports/tr11
[Jamo]	Data file ftp://ftp.unicode.org/Public/UNIDATA/Jamo.txt
[LineBreak]	Data file ftp://ftp.unicode.org/Public/UNIDATA/LineBreak.txt
[Line Breaking]	Asmus Freytag, Unicode Standard Annex #14: Line Breaking Properties, http://www.unicode.org/unicode/reports/tr14
[NamesList]	Data file ftp://ftp.unicode.org/Public/UNIDATA/NamesList.txt
[NamesList-Format]	Readme file http://www.unicode.org/Public/UNIDATA/NamesList.html
[SpecialCasing]	Data file ftp://ftp.unicode.org/Public/UNIDATA/SpecialCasing.txt

[Unicode]	The Unicode Standard, Version 3.0, Addison Wesley Longman, 2000.
[UnicodeCharacterDatabase]	Overview file, http://www.unicode.org/UCD/
[UnicodeData]	Data file ftp://ftp.unicode.org/Public/UNIDATA/UnicodeData.txt
[UnicodeData-Format]	Readme file http://www.unicode.org/Public/UNIDATA/UnicodeData.html
[UnicodeVersions]	Versions of the Unicode Standard http://www.unicode.org/unicode/standard/versions

Acknowledgements

The author wishes to thank Ken Whistler and Mark Davis for their insightful comments.

Changes from previous drafts

1 First version for public review

Copyright © 2000–2002 Unicode, Inc. All Rights Reserved. The Unicode Consortium makes no expressed or implied warranty of any kind, and assumes no liability for errors or omissions. No liability is assumed for incidental and consequential damages in connection with or arising out of the use of the information or programs contained or accompanying this technical report.

Unicode and the Unicode logo are trademarks of Unicode, Inc., and are registered in some jurisdictions.