Technical Reports

L2/15-038

Proposed Update

Unicode Technical Report #23

THE UNICODE CHARACTER PROPERTY MODEL

~~Author~~ Editors	Ken Whistler (ken@unicode.org) and Asmus Freytag (asmus@unicode.org)
Date	2014-11-17
This Version	http://www.unicode.org/reports/tr23/tr23-10.html
Previous Version	http://www.unicode.org/reports/tr23/tr23-9.html
Latest Version	http://www.unicode.org/reports/tr23/
Revision	10

Summary

This document presents a conceptual model of character properties defined in the Unicode Standard.

Status

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 Code Point Properties and Abstract Character Properties
   2.5 Normative Properties
   2.6 Informative Properties
   2.7 Referring to Properties
   2.8 The Unicode Character Database
3. Definitions
   3.1 Properties and Property Values
   3.2 Types of Property Values
   3.3 Types of Properties
   3.4 Conformance Status of Properties
   3.5 Classification of Properties
   3.6 String Functions
   3.7 Classification of String Functions
   3.8 Other Definitions
4. Conformance-related Considerations
   4.1 Conformance Requirements
   4.2 Algorithms and Character Properties
   4.3 Overriding Properties and Higher-level Protocols
5. Updating Character Properties and Extending the Standard
   5.1 Updating Properties
   5.2 Stability Guarantees
   5.3 Consistency of Properties
   5.4 Provisional Properties
   5.5 Stabilized Properties
6. Special Property Values
   6.1 Not Applicable Value
   6.2 Default Values
   6.3 Preliminary Property Assignments
References
Acknowledgements
Modifications

1. Scope

This report discusses of common aspects of character properties. This description of the Unicode character property model is not intended to supersede the normative information on properties in The Unicode Standard [Unicode], nor the existing body of technical reports and documentation files in the Unicode Character Database [UCDDoc] that provide detailed descriptions for particular character properties. Instead it presents a general overview and typology of character properties and property values.

~~In some ways, the model of character properties presented here goes beyond the current text of the standard, because it lays the foundation for a future clarification of the definition of character properties in later updates to the Unicode standard.~~

This report specifically covers formal **character properties**, which are those attributes of characters specified according to the definitions set forth in this report. Such formal character properties are only a subset of character properties in the generic sense, and they further subdivide into those properties defined in the Unicode Standard or Unicode Character Database, and those defined by related standards.

2. Overview

At its most basic, a character property relates a character to a value. At its most general, a property can be considered a function that maps from code points to specific property values.

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.

> *D3 Character semantics:* The semantics of a character are determined by its identity, normative properties, and behavior.

The assignment of character semantics in the Unicode Standard is based on character behavior. Other character set standards leave it 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 character behavior.

2.2 Character Behavior in Context

The interpretation of some properties (such as whether a character is a digit or not) 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 title casing, and the classification of neutrals in script assignments. The line breaking rules of *UAX#14 Unicode Line Breaking Algorithm* [LineBreak] involve character pairs 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.

In some cases, the expected character behavior depends on external context, such as the type and nature of the document, the language of the text, or the cultural expectations of the user. Properties modeling such behaviors may be specified in separate standards, as is the case for the *UTS#10 Unicode Collation Algorithm* [UCA]. Where a reasonably generic set of property values can be assigned, for example for [LineBreak], such properties may be defined as part of [Unicode]. Such properties and any algorithms related to them define useful default behavior, which can be further customized or tailored to meet more specific requirements.

2.3 Relation of Character Properties to Algorithms

When modeling character behavior with computer processes, formal character properties are assigned to achieve the expected results. Such modeling depends heavily on the algorithms used to produce these results. 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 that can make use of the same general character property. The last case may require occasional implementation-specific adjustments in character property assignment to make all algorithms work correctly. This can usually be achieved by overriding specific properties for specific algorithms. (See also Section 4.3 "Overriding Properties via Higher-level Protocols")

When assigning character properties for use with a given algorithm, it may be tempting to assign somewhat arbitrary 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 essential underlying linguistic identity of the character. However, not all aspects of a character's identity are relevant in all circumstances, and some characters can be used in many different ways, depending on context or circumstance. This means the formal character properties alone are not sufficient to describe the complete range of desirable or acceptable character behaviors.

> **Note:** In some cases, the relevant algorithm is not defined in the Unicode standard. For example, the algorithm that converts strings of digits into numerical values is not defined in the Unicode Standard, but implementations will nevertheless refer to the numeric_value property.

2.4 Code Point And Abstract Character Properties

Code point properties are properties of code points per se: in a character encoding standard these are independent of any assignment of actual abstract characters to those code points. In most character encoding standards, these are trivial, but in the Unicode Standard they are not.

Examples of code point properties include:

- Code point XXX is a surrogate code point.
- Code point XXX is a private use code point.
- Code point XXX is a reserved code point.
- Code point XXX is reserved for encoding format control characters.
- Code point XXX is earmarked for encoding a RTL script.
- Code point XXX is a Pattern_Syntax code point.
- Code point XXX is a Pattern_Whitespace code point.
- Code point XXX is located on Plane 1.

These statements remain true of a code point whether or not there is a particular abstract character assigned to them. ~~, or~~ For example, they track status of the code points: whether any abstract character is assigned to them or can be assigned to them, and so on. Essentially, whenever code points are designated or ranges are reserved in some way, ~~by constraining what they can contain in the future,~~ code point properties are assigned.

Character properties are those properties that abstract characters have independent of any consideration of their encoding.

Examples of character properties, not limited to formal properties, include:

- G is an alphabetic character.
- G is in the Latin script. ~~G is an uppercase letter.~~
- G is an uppercase letter.
- G is not used in hexadecimal expressions.
- G collates after F in the English alphabet.
- G was putatively invented by Spurius Carvilus Ruga ca. 300.
- G commonly represents the velar voiced stop in orthographies.
- G is not a punctuation character.
- G denotes giga in the SI system of nomenclature.
- G has no diacritic.
- G is a base character.
- G is not a combining character.

By virtue of encoding the abstract character LATIN CAPITAL LETTER G at the code point U+0047, this universe of character properties, some known and obvious, others obscure or even undiscovered, are associated with that code point.

Some of those character properties are generic and systematic enough to be useful or even necessary in the implementation of general text processing algorithms — those are the ones that the Unicode Standard formalizes as properties in the Unicode Character Database.

General text processing algorithms and the programming APIs through which they are accessed, must be prepared to deal with any code point, even one that is unassigned to any characters at the time the implementation was created. As a result, they nearly always need to properly handle each and every code point for any character property, even if they only associate a property value of 'unknown' or 'inapplicable' to unassigned or unsupported code points.

This requirement leads to the use of the unifying concept of **Encoded Character Property** in the Unicode character property model. An encoded character property combines the concept of a code point property associating ranges of code points with default values of a property, with the concept of a character property associating specific values to the assigned characters. This unified model correlates well with the reality of Unicode-based implementations, which must supply some value for each and every code point. In addition, this unified concept simplifies most of the definitions that are built on top of it, since it is no longer necessary to separately account for definitions applying to character properties vs. code point properties.

2.5 Normative Properties

In Chapter 3, *Conformance*, The Unicode Standard [Unicode] defines a *Normative Property* as "a Unicode character property used in the specification of the standard" (definition *D33*) and provides the following explanation:

> Specification that a character property is normative means that implementations which claim conformance to a particular version of the Unicode Standard and which make use of that particular property must follow the specifications of the standard for that property for the implementation to be conformant. For example, the Bidi_Class property ~~directionality property (bidirectional character type)~~ is required for conformance whenever rendering text that requires bidirectional layout, such as Arabic or Hebrew.
>
> Whenever a normative process depends on a property in a specified way, that property is designated as normative.
>
> The fact that a given Unicode character property is normative does not mean that the values of the property will never change for particular characters. Corrections and extensions to the standard in the future may require minor changes to normative values, even though the Unicode Technical Committee strives to minimize such changes...
>
> Some of the normative Unicode algorithms depend critically on particular property values for their behavior. Normalization, for example, defines an aspect of textual interoperability that many applications rely on to be absolutely stable. As a result, some of the normative properties disallow any kind of overriding by higher-level protocols. Thus the decomposition of Unicode characters is both normative and *not overridable*; no higher-level protocol may override these values, because to do so would result in non-interoperable results for the normalization of Unicode text. Other normative properties, such as case mapping, are *overridable* by higher-level protocols, because their intent is to provide a common basis for behavior. Nevertheless, they may require tailoring for particular local cultural conventions or particular implementations.

By making a property normative and non-overridable, the Unicode Standard guarantees that conformant implementations can rely on other conformant implementations to 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, such as the decompositions and their use by the Normalization forms [Normal].

> **Note**: One trivial, but important example of conformant implementation is runtime access to information from the Unicode Character Database [UCD]. For normative properties exposed by a conformant implementation, conformance requires the returned values to match the values defined by the Unicode Consortium.

For some character properties, such as the general category, the Unicode standard does not define what model of processing the property is intended to support, nor does it specify the required consequences of a character being defined as "Letter Other" as opposed to "Symbol Other", for example. In the absence of such definition, the only effect of conformance that can be rigorously tested is whether a conformant implementation of a character property function returns the correct value to its caller. However, many implementations use such normative properties for their own purposes and guaranteed access to this information helps interoperability.

For information on which properties are normative, see the documentation file for the Unicode Character Database [UCDDoc].

For more information on overriding normative properties, see Section 4.3 *Overriding properties via Higher-level Protocols*.

2.6 Informative Properties

The Unicode Standard [Unicode] defines an *Informative Property* as "a Unicode character property whose values are provided for information only" (definition *D35*) and provides the following explanation:

> ~~*D35 Informative property:* A Unicode character property whose values are provided for information only.~~
>
> A conformant implementation is free to use or change ~~such~~ informative property values as it may require, while remaining conformant to the standard. An implementer has the option of establishing a protocol to convey that particular informative properties are being used in distinct ways.
>
> Informative properties capture expert implementation experience. When an informative property is explicitly specified in the Unicode Character Database, its use is strongly *recommended* for implementations to encourage comparable behavior between implementations. Note that it is possible for an informative property in one version of the Unicode Standard to become a normative property in a subsequent version of the standard if its use starts to acquire conformance implications in some part of the standard. [emphasis added].

Properties may be informative for two main reasons:

1. The exact nature or applicability of the property may be unclear. In some cases, the precise set of characters to which it applies may also not be well-determined.
2. Existing implementations show a range of behaviors for the same character, many or all of which may be equally useful choices on the part of their designers.

In some cases, properties are too tentative to be published as informative properties. In that case they may be explicitly designated as *provisional*.

2.7 Referring to Properties

The Property Aliases [Alias] and Property Value Aliases [ValueAlias] define a set of names and abbreviations, called *aliases*, that are used to refer to properties and property values. These names can be used for XML formats of data in the Unicode Character Database [UCD], for regular-expression property tests, and other programmatic textual descriptions of Unicode data. The names themselves are not normative, except where they correspond to normative properties in the UCD. However, other standards may make normative references to both normative and informative aliases. For more information, see UTS #18: *Unicode Regular Expressions* [RegEx].

There is one abbreviated name and one long name for most of the properties. Additional aliases may be added at any time. The property *value* names are *not* unique across properties. For example, **AL** means Arabic Letter for the Bidi_Class property, and **AL** means Alpha_Left for the Combining_Class property, and **AL** means Alphabetic for the Line_Break property. In addition, some property names may be the same as some property value names. For example, **cc** means Combining_Class property, and **cc** means the General_Category property value Control. The combination of property value and property name is, however, unique.

The aliases may be translated in appropriate environments, and additional aliases may be used. The case distinctions, whitespace, and '_' in the property names are not normative. Unless a specific form is required in a particular application, all forms are equivalent. For further information see section 5.9 Matching Rules in UAX#44 Unicode Character Database [UCDDoc].

[Unicode] Section 3.1 gives a prescription for referencing properties:

References to Unicode Character Properties

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

> Property: General_Category (gc)
> Property Value: Uppercase_Letter (Lu)

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

> The property value Uppercase_Letter from the General_Category property, as defined in Unicode 7.0.0

Then cite that version of the standard, using the standard citation format that is provided for each version of the Unicode Standard. For Unicode 7.0.0, it is:

> The Unicode Consortium. The Unicode Standard, Version 7.0.0, (Mountain View, CA: The Unicode Consortium, 2014. ISBN 978-1-936213-09-2)
> http://www.unicode.org/versions/Unicode7.0.0/

Additional reference examples are available online.

2.8 The Unicode Character Database

The Unicode Character Database [UCD] is the main repository for machine-readable character properties. It consists of a number of files containing property data along with a documentation file explaining the organization of the database and the format and meaning of the property data. The main file, "The Unicode Character Database" [UCDDoc] explains the overall organization of the current version of the UCD and tells which files contain which properties.

While the Unicode Consortium strives to minimize changes to character property data, occasionally the character properties for already encoded characters must be updated. When this situation occurs, the relevant data files of the Unicode Character Database are revised. The revised data files are posted on the Unicode Web site as an update version of the standard.

A visual documentation of character code point, character name and reference glyph, together with excerpts from some of the character properties and augmented by additional annotations can be found in the Character Code [Charts].

3. Definitions

The following presents a consistent set of definitions related to character properties. Where possible, these definitions match the formal definitions in Chapter 3, Conformance, in [Unicode]. In those cases, the original number of the definition is given at the end in square brackets. As much as possible, the definition numbers in this document will be retained as new definitions are added. When referring to these definitions in other contexts, it is customary to prefix the term 'Unicode' to the defined term to indicate the context. For example 'Character Property', becomes 'Unicode Character Property', etc.

3.1 Properties and Property Values

PD1. Property
A named attribute of an entity in the Unicode Standard, associated with a defined set of values. [D19]

PD2. Code Point Property
A property of code points. [D20]

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.

PD3. Abstract Character Property
A property of abstract characters. [D21]

PD4. Encoded character property.
A property of encoded characters in the Unicode Standard. [D22]

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

Encoded character properties typically map a default value to any code point not assigned to a character.

In the rest of this document, as in the Unicode Standard, the term 'character property', or the term 'property' without qualifier includes both character and code point properties and their combined form, the encoded character properties.

PD5. Property Value
One of the set of values associated with a property. [D23 - but there limited to 'encoded character property']

For example, the East Asian Width [EAW] property has the possible values "Narrow", "Neutral", "Wide", "Ambiguous" and "Unassigned". See [Alias] and [ValueAlias] for a list of labels for properties and their values respectively.

3.2 Types of Property Values

PD6. Explicit Property Value
A value for and encoded character property which is explicitly associated with a code point in one of the data files of the Unicode Character Database. [D24].

PD7. Implicit Property Value
A value for an encoded character property which is given by a generic rule or by an "otherwise" clause in one of the data files of the Unicode Character Database [D25].

PD8. Default Property Value
~~For a given code point property, any value of that property which is assigned, by default, to unassigned code points or to code points not explicitly specified to have other values of that property.~~ The value (or in some cases small set of values) of a property associated with unassigned code points or with encoded characters for which the property is irrelevant.[D26]

Note: There may be more than one default value per property, with different values for different ranges, as in the Bidi property.

3.3 Types of Properties

PD9. Enumerated Property
A property with a small set of named values. [D27]

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

PD10. Closed Enumeration
An enumerated property for which the set of values is closed and will not be extended for future versions of the Unicode Standard. [D28]

Note: Currently, the General_Category and Bidi_Class properties are the only closed enumerations, other than Boolean properties.

PD11. Boolean Property
A closed enumerated property whose set of values is limited to 'true' and 'false'. [D29]

The presence or absence of the property is the essential information.

A Boolean property is sometimes called a 'single valued' property since 'false' often has the meaning of 'this property does not apply'.

PD12. Numeric Property
A numeric property is a property whose value is a number that can take on any integer, or real value. [D30]

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

PD13. String-Valued Property

A property whose value is a string. [D31]

The Canonical_Decomposition property is a string-valued property.

PD14. Catalog property

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

Examples are the Age, ~~and~~ Block, and Script properties. Additional new values may be added to the set of enumerated values each time the standard is revised. ~~Additional values for may be added each time the standard is revised.~~ Each new Unicode version adds a new value for Age. When a new block is added to the standard, a corresponding new value is added to the Block property. Likewise, when a new script is added, a corresponding new value of the Script property is also added.

PD15. Miscellaneous property

A property whose values are not Boolean, enumerated, numeric, string or catalog values.

The Unicode character name property is a miscellaneous property.

3.4 Conformance Status of Properties

PD16. Normative Property

A [Unicode character] property used in the specification of the Unicode standard. [D33]

Note: A normative process that depends on a property in a normative and testable way, is usually sufficient reason to designate a property as normative. For example, the interpretation of the *bidirectional class* is precisely defined in [Bidi].

If a process does not interpret a given character, it may remain unaware of its properties. However, it is recommended that processes use carefully-chosen default values for characters that they do not handle.~~S~~

See also Section 2.5, Normative Properties)

PD17. Overridable Property

A normative property whose values may be overridden by conformant higher-level protocols. [D34]

See Section 4.3 *Overriding properties via Higher-level Protocols*.

PD18. Informative Property

A [Unicode character] property whose values are provided for information only. [D35]

Note: Informative properties capture expert implementation experience and their use is strongly recommended by the Consortium, but there are no requirements on implementations of the Unicode Standard.

See also Section 2.6, Informative Properties)

PD19. Provisional Property

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

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

See also Section 5.4, Provisonal Properties)

3.5 Classification of Properties

The following definitions do not define character or code point properties, but properties of such properties. In the definitions in this section, the term 'code point' is used inclusively to mean code point for a code point property and character for a character property, respectively.

PD20. Context-dependent Property

A property that applies to a code point in the context of a longer code point sequence. [D37]

For example, the lower case mapping of Greek sigma depends on the surrounding characters. See also PD33: *Context-dependent String Function.*

PD21. Context-independent Property

A property that is not context-dependent: it applies to a code point in isolation. [D38]

PD22. Stable Transformation

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

PD23. Stable Property

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

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, the Canonical Combining Class, while not immutable, is a stable property with respect to the Normalization Forms as defined in [Normal].

Note: As new characters are assigned to previously unassigned code points, replacing any default values for these code points with actual property values must maintain stability.

PD24. Fixed Property

A property whose values, (other than the default value) once associated with a character or other designated code point, are fixed and will not be changed, except to correct obvious or clerical errors. [D41].

For a fixed property, any default values can be replaced without restriction by actual property values, as new characters are assigned to previously unassigned code points. Examples of fixed properties are Age or Hangul Syllable Type.

Note: Designating a property as fixed does not imply stability or immutability, see below. While the age of a character, for example, is established by the version of the Unicode Standard at which it was added, errors in the published listing of the property value could be corrected. For some other properties, there are explicit stability guarantees that prohibit the correction even of such errors. See Section 5.2 *Stability Guarantees*.

PD25. Immutable Property

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

An immutable property is trivially stable with respect to *all* ~~context-free~~ algorithms. [Note: The text of TUS and TR differ for no apparent reason.] An example of an immutable property is the Unicode character name. See Section 5.2 *Stability Guarantees*.

Note: Because character names are values of an immutable property, misspellings and incorrect names will *never* be corrected. Any errata will be noted in a comment in the names list, and, where needed, an informative character name alias will be provided.

PD26. Stabilized Property

A property which is neither extended to new characters, nor maintained in any other manner, but which is retained in the Unicode Character Database. [D43]

A stabilized property is also a fixed property.

PD27. Deprecated Property

A property whose use by implementations is discouraged. [D44]

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

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

PD28. Simple Property

A property whose values are specified directly in the Unicode Character Database (or elsewhere in the Unicode Standard) and whose values cannot be derived from other simple properties. [D45]

PD29. Derived Property

A property whose values are algorithmically derived from some combination of simple properties. [D46]

PD30. Property Alias

A unique identifier for a particular [Unicode character] property. [D47]

The set of property aliases forms a namespace. See Section 2.7 Referring to Properties.

PD31. Property Value Alias

A unique identifier for a particular enumerated value for a particular [Unicode character] property. [D48]

The set of property value aliases for each property form a separate namespace. Values from different properties may have non-unique names. As a trivial example, the property value aliases for all Boolean properties are 'true' and 'false'.

See also Section 2.7 Referring to Properties.

3.6 String Functions

None of the following definitions is found in the Unicode Standard at this point, however, they are useful in the context of discussing Unicode algorithms and their relation to properties.

PD32. String

An ordered sequence of zero or more code units.

This ~~is related to~~extends the definition of *code unit sequence* [D78], ~~but also~~ to allow~~s~~ the empty string. In addition, unlike the definition of Unicode String~~s~~ [D80], a string as defined here is not limited to Unicode code units. Character mappings are common examples of properties for which the values are strings but not Unicode Strings.

PD33. Offset

An offset into a string is a number from 0 to *n* where *n* is the length of the string in code units~~, and~~ It indicates a position that is logically ~~adjacent~~ between ~~Unicode~~ code units. An offset of 0 indicates the position before the first code unit in the string, and an offset of *n* indicates the position after the last code unit in the string.

PD34. Code Point Aligned Offset

An offset into a string that is aligned to a code point boundary.

PD35. String Function

A string function is a function whose input is a string *S* and two offsets *a* and *b*, with $a \le b$.

PD36. Text Boundary Property

A string function whose value is defined for a particular offset.

Text boundary functions are also called segmentation functions, because they are commonly used to return segments of text between boundaries. A simple text boundary function, like IsBreak(S,a,b) minimally returns a Boolean value. However, other text boundary functions may return additional information. For example, a word-selection boundary function may return whether the previous segment contained a letter, or a linebreak function may return information on the relative priority of the break.

3.7 Classification of String Functions

PD37. Context-independent String Function

Given a string *S*, and offsets *a* and *b*, a context-independent string function is any string function *F* for which *F*(*S,a,b*) is independent of the content of *S* before *a* and after *b*.

In other words, the input to a context-independent function is fully defined by the code points between the given offsets.

PD38. Context-dependent String Function

A context-dependent string function is a string function that is not context-independent.

In other words, the input to a context-dependent string function requires additional information, such as information about the code points surrounding the code point range defined by the offsets as well as the code points in the range. Any text boundary function of the form *B* (*S,x,x*) is by definition context dependent.

PD39. String Transform

A string-valued string function.

PD40. Idempotent String Function (Folding)

A string transform *F*, with the property that repeated applications of the same function *F* produce the same output: *F*(*F*(*S*)) = *F*(*S*) for all input strings *S*.

Such a string function is also called a folding.

A folding establishes an equivalence relation, whereby X ≡ Y if and only if F(X) = F(Y). This equivalence relation partitions the set of all strings into the set of equivalence classes for the relation. Conversely, any partition of strings can be used to generate a folding, by choosing one element of each partition to be the "target member" that the members of that partition map to.

The notation toX(s) may be used for the folding, and isX(s) for the corresponding binary function, defined such that isX(s) if and only if toX(s) = s. For example, toNFC() is the folding that converts to NFC format, while isNFC() is the test for whether a string is in that format.

A well known example of a folding function is case folding. For case folding, the equivalence class consists of all case variations, including upper, lower, title case and mixed case. In the case of Unicode case folding, the target member is chosen to be the lowercase character.

Folding functions may be context dependent. Normalization is an example of a context dependent folding.

PD41. Code Point Count Preserving String Function

A string function whose result is a string containing the same number of code *points* as its input, is a count preserving string function.

PD42. Buffer Length Preserving String Function

A string function whose result is a string containing the same number of code *units* as its input, is a buffer length preserving string function.

3.8 Other Definitions

PD43. Higher-level Protocol

Any agreement on the interpretation of Unicode characters that extends beyond the scope of ~~this~~the Unicode Standard. [D16] [This is a case where the definition text must be slightly reworded due to context.]

4. Conformance-related Considerations

This technical report does not define conformance requirements, but the following subsections discuss and summarize the conformance requirements related to character properties stated in the Unicode Standard. Where applicable, the number of the corresponding conformance clause or definition is given in square brackets.

4.1 Conformance Requirements

In Chapter 3, Conformance, The Unicode Standard [Unicode] states that *"A process shall interpret a coded character sequence according to the character semantics established by this standard, if that process does interpret that coded character sequence." [C4]* 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 Algorithms and Character Properties

Unicode algorithms are specified as an idealized series of steps (rules) performed on an input of character codes and their associated properties. [Unicode] states:

- An implementation claiming conformance to a Unicode algorithm need only guarantee that it produces the same results as those specified in the logical description of the process; it is not required to follow the actual described procedure in detail. This allows room for alternative strategies and optimizations in implementation. See [C18]

As long as the same results are achieved, the implementation is also not required to use the actual properties published in the [UCD]. *Overriding* a property value therefore does not necessarily imply an actual change in property assignments, merely that the conformant implementation of an algorithm now produces the same results as if the property values had been changed in the description of the ideal algorithm.

4.3 Overriding Properties via Higher-level Protocols

In discussing character semantics, the Unicode Standard [Unicode] makes this statement about overriding properties and character behavior:

> Some normative behavior is default behavior; this behavior can be overridden by higher-level protocols. However, in the absence of such protocols, the behavior must be observed so as to follow the character semantics. See [D3]

Overrides by a higher-level protocol can conceptually take many forms, including, but not limited to:

- providing artificial context for an algorithm that defines a context-dependent string function
- applying the algorithm on a substring
- emulating the effect of format control characters in markup
- reassigning a different property value to a character during processing or rendering
- changing the result of a string function for particular inputs

Where overrides involve normative properties, specific restrictions apply, for example:

> • The character combination properties and the canonical ordering behavior cannot be overridden by higher-level protocols. See [D3]

For additional examples of higher-level protocols as well as restrictions on them see section 4.3 in UAX #9: *Unicode Bidirectional Algorithm* [Bidi]. There are some normative properties that are fully overridable, for example General Category.

On the other hand, any and all informative properties may be overridden. However, if doing so changes the result of a Unicode Algorithm, any implementation wishing to conform to that algorithm must indicate that overrides have been applied.

5. Updating Properties and Extending the Standard

5.1 Updating Properties

Updates to properties 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

While the Unicode Consortium endeavors to keep the values of all character properties as stable as possible, some circumstances may arise that require changing them. Changing a character's property assignment may impact existing implementations and is therefore done judiciously and with great care, only when there is no better alternative.

In particular, as Unicode encodes less well-documented scripts, such as those for minority languages, the exact character properties and behavior may not be known when the script is first encoded. The properties for such characters are expected to be changed as information becomes available.

As implementation experience grows, it may become necessary to readjust property values. As much as possible, such readjustments are compatible with established practice. Occasionally, a character property is changed to prevent incorrect generalizations of 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 is now classified as a Formal Control (gc=Cf) to distinguish this line break control from space characters.

In other cases, there may have been unintentional mistakes in the original information that require corrections.

The [UTC] carefully weighs the costs of a change against the benefit of the correction. In addition, all updates to properties are subject to the stability guarantees described in the next section.

5.2 Stability Guarantees

Unicode guarantees the stability of character assignments; that is, the *identity* of a character encoded at a given location will remain the same. Once a character is encoded, its properties may still be changed, but *not* in such a way as to change the fundamental identity of the character.

For example, the representative glyph for U+0041 "A" could not be changed to "B"; the general category for U+0041 "A" could not be changed to Ll *(lowercase letter);* and the decomposition mapping for U+00C1 (Á) could not be changed to <U+0042, U+0301> (B, ´).

In addition, for some properties, one or more of the following aspects are guaranteed to be invariant:

- stability of assignment
- stability of result when applying the property
- stability of set of values for a property
- stability of relation to another property
- stability of file formats

For the most up-to-date specification of all stability guarantees in effect see the Unicode Character Encoding Stability Policy [Stability]. Note that the status of a property as normative does not imply a stability guarantee.

5.2.1 Stability of Assignment

Stability of assignment is the characteristic of an *immutable* property. For example, once a character is encoded, its code point and name are immutable properties. An immutable property~~ies~~ allows software and documents to refer to its values without needing to track future updates to the Standard. One side effect of an immutable property is that errors in property values cannot be fixed. For example, mistakes in naming are annotated in the Unicode character names list in a note or by using an alias, but the formal name remains unchanged, even in cases of clear-cut typographical errors.

Because Code_Point is an immutable property, if a character is ever found to be unnecessary, or a mistaken duplicate of an existing character, it will not be removed. Instead, it can be given an additional property, *deprecated*, and its use strongly discouraged. However, the interpretation of all existing documents containing the character remains the same.

5.2.2 Stability of Result when Applying the Property

Stability of result is the characteristic of a *stable* property. For example, once a character is encoded, its canonical combining class and decomposition (canonical or compatibility) are stable with respect to normalization. Stability with respect to normalization is defined in such a way that if a string contains only characters from a given version of the Unicode Standard (say Unicode 3.2), and it is put into a normalized form in accordance with that version of Unicode, then it will be in normalized form when normalized according to any future version of Unicode.

However, unlike character code and character name, some properties that are guaranteed to be stable may be corrected in *exceptional* circumstances that are clearly defined by the Unicode Character Encoding Stability Policy [Stability]. In addition to other requirements, the correction must be of an obvious mistake, such as a typographical error, and any alternative would have to violate the stability of the identity of the character in question. Allowing such carefully restricted exceptions obviates the need for encoding duplicate characters simply to correct clerical or other clear-cut errors in property assignments.

5.2.3 Stability of Set of Values for a Property

For most properties, additional property values may be created and assigned to both new and existing characters. For example additional line breaking classes will be assigned if characters are discovered to require line breaking behavior that cannot be expressed with the existing set of classes. For other properties the set of values is guaranteed to be fixed, or their range is limited. For example, the set of values for the General Category or Bidirectional Class is fixed, while Combining classes are limited to the values 0 to 255.

5.2.4 Stability of Relation to Another Property

In many cases, once a character has a certain value for one property, it is likely to have a particular value for a given other property. These relations are used by the Unicode

Consortium in assigning properties to new characters, and in evaluating properties for internal consistency. In some cases, such dependencies are explicitly guaranteed and stable.

For example, all characters other than those of General Category M* have the combining class 0.

5.2.5 Stability of File Formats

In principle, the way the property information is presented in the Unicode Character Database is independent of the way this information is defined. However, as the Unicode Standard gets updated, it becomes easier for implementations to track updates if file formats remain unchanged and other aspects of the way the data are organized can remain stable. For the majority of properties, such stability is an informal goal of the development process, but in a few cases, some aspects of the data organization are covered by formal stability guarantees.

For example, Canonical and Compatibility mappings are always in canonical order, and the resulting recursive decomposition will also be in canonical order. Canonical mappings are also always limited either to a single value or to a pair. The second character in the pair cannot itself have a canonical mapping.

As an alternative to the legacy conventions of semicolon-separated text files, the Unicode Character Database is now also available as a single XML file. See UAX#42 Unicode Character Database in XML [XML].

5.3 Consistency of Properties

In an ideal world, all character properties would be perfectly self-consistent, and related properties would be consistent with each other over the entire range of code points. However, The Unicode Standard is the product of many compromises. It has to strike a balance between uniformity of treatment for similar characters, and compatibility with existing practice for characters inherited from legacy encodings. Because of this balancing act, one can expect a certain number of anomalies in character properties.

Sometimes it may be advantageous for an implementation to purposefully override some of the anomalous property values, increasing the efficiency and uniformity of algorithms—as long as the results they produce do not conflict with those specified by the normative properties of this standard. See Chapter 4, *Character Properties* in [Unicode] for some examples.

Property values assigned to new characters added to the Unicode Standard are generally defined so that related characters are given consistent values, unless deliberate exceptions are needed. For some properties, definite links between that property and one or more other properties are defined. For example, for the LineBreak property, many line break classes are defined in relation to General Category values.

There are some properties that are interrelated or that are derived from a combination of other properties, with or without a list of explicit exceptions. When properties are assigned to newly assigned characters, or when properties are adjusted, it is necessary to take into account all existing relevant properties, any derivational relations to derived properties, and all property stability guarantees.

5.4 Provisional Properties

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

5.5 Stabilized Properties

Occasionally, as the standard matures, and new characters, properties or algorithms are defined, the information presented in an existing property may be better represented via other properties, or it may no longer make sense to extend the property to new characters. Such a property may then no longer be maintained in future versions of the Unicode Standard. In that case, it will be designated as *stabilized*. For backwards compatibility, a stabilized property will remain part of the Unicode Character database, but will not be updated or corrected.

An example of a stabilized property is Hyphen.

6. Special Property Values

6.1 Not Applicable Value

Limited properties apply to only a subset of characters. Where these properties are implemented as a partition of the Unicode code space, the characters to which the property does not apply are given a special value denoting that the property does not apply. The "not applicable" value may be the explicit value "N/A" or, for some properties, take other values such as "XX".

6.2 Default Values

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. For details of these recommendations for various

properties see [UCDDoc].

2. Treat the unassigned areas of a given character block as if they had property values common to other characters of the block. A variation of this scheme bridges small gaps in the allocation inside a block by using the property values for the characters bracketing the hole.
3. Give an unassigned *code point* an implementation defined default property that will result in graceful if not completely correct behavior, if an encoded characters is later assigned at that code point.

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 more gracefully in such circumstances.

In principle, default values are temporary: they are superseded by final assignments once characters are assigned to a given code point.

For noncharacter code points, a character property function would return the same value as the default value for unassigned characters.

6.3 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.

This is distinct from provisional properties, where the entire property is preliminary.

References

[Alias] Property Aliases
http://www.unicode.org/unicode/Public/UNIDATA/PropertyAliases.txt

[Bidi] Unicode Standard Annex #9: Unicode Bidirectional Algorithm
http://www.unicode.org/reports/tr9/

[Charts] The online code charts can be found at http://www.unicode.org/charts/ An index to characters names with links to the corresponding chart is found at http://www.unicode.org/charts/charindex.html

[EAW] Unicode Standard Annex #11: East Asian Width
http://www.unicode.org/reports/tr11/

[Feedback] Reporting Errors and Requesting Information Online
http://www.unicode.org/reporting.html

[FAQ] Unicode Frequently Asked Questions
http://www.unicode.org/faq/
For answers to common questions on technical issues.

[Glossary] Unicode Glossary
http://www.unicode.org/glossary/
For explanations of terminology used in this and other documents.

[LineBreak] Unicode Standard Annex #14: Unicode Line Breaking Algorithm
http://www.unicode.org/reports/tr14/

[Normal] Unicode Technical Report #15: Unicode Normalization Forms
http://www.unicode.org/unicode/reports/tr15/

[RegEx] Unicode Technical Standard #18: Unicode Regular Expressions http://www.unicode.org/unicode/reports/tr18/

[Reports] Unicode Technical Reports
http://www.unicode.org/reports/
For information on the status and development process for technical reports, and for a list of technical reports.

[Stability] Unicode Character Encoding Stability Policy
http://www.unicode.org/policies/stability_policy.html

[UCA] Unicode Technical Standard #10: Unicode Collation Algorithm
http://www.unicode.org/reports/tr10/

[UCD] About the Unicode Character Database.
http://www.unicode.org/ucd/
For an overview of the Unicode Character Database

[UCDDoc] Unicode Character Database.
http://www.unicode.org/reports/tr44/
For documentation of the contents of the Unicode Character Database and its associated files

[Unicode] The Unicode Standard
For the latest version see: http://www.unicode.org/versions

	/latest/. For the last major version see: The Unicode Consortium. The Unicode Standard, Version 7.0. (Mountain View, CA: The Unicode Consortium, 2014. ISBN 978-1-936213-09-2).
[Unihan]	The Unihan Database. For character information about CJK ideographs; for more information about the database see [UCDDoc]. The database itself is available online at http://www.unicode.org/Public/UNIDATA/Unihan.zip (6 MB)
[UTC]	The Unicode Technical Committee, for more information see http://www.unicode.org/consortium/utc.html
[ValueAlias]	Property Value Aliases http://www.unicode.org/Public/UNIDATA/PropertyValueAliases.txt
[Versions]	Versions of the Unicode Standard http://www.unicode.org/standard/versions/ For information on version numbering, and citing and referencing the Unicode Standard, the Unicode Character Database, and Unicode Technical Reports.
[XML]	Unicode Character Database in XML http://www.unicode.org/reports/tr42/ The XML version of the database is available online at http://www.unicode.org/Public/UCD/latest/ucdxml/

Acknowledgements

Asmus Freytag was the initial author of this report, with additional content provided by Ken Whistler.

The authors wish to thank Mark Davis for his extensive contributions and insightful comments, and Dr. Julie Allen for extensive copy-editing.

Modifications

The following summarizes modifications from the previous version of this document.

Revision 10 [AF, KW]

- Proposed update of the report.
- Updated references to and citations from Unicode 7.0.
- Minor editing.

Revision 9 [KW]

- Added a note on constraints on new property additions.
- Updated titles of some references. Other minor edits.
- Reformatted modifications section to follow bulleted UTR style.
- Dropped irrelevant early draft modification information preceding the first approved version.
- Added Ken Whistler as author.

Revision 8 being a proposed update, only modifications between revisions 7 and 9 are noted here.

Revision 7 [AF]

- Removed definition PD30 (Limited Property).
- Updated all definitions that correspond to definitions in Unicode 5.0.0.
- Updated all definition numbers according to 5.0.0.
- New section 2.4.
- Additional updates to the text and some other definitions.

Revision 6 being a proposed update, only modifications between revisions 5 and 7 are noted here.

Revision 5 [AF]

- Initial approved version of this report.