Technical Reports

L2/03-200

Draft Unicode Technical Report #23

THE UNICODE CHARACTER PROPERTY MODEL

Version	1.0
Author	Asmus Freytag (asmus@unicode.org)
Date	2003-06-10 - UTC REVIEW DRAFT
This Version	http://www.unicode.org/reports/tr23/tr23-3.html
Previous Version	http://www.unicode.org/reports/tr23/tr23-2.html
Latest Version	http://www.unicode.org/reports/tr23/
Tracking Number	3

Summary

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

Status

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 This is a draft document which may be updated, replaced, or superseded by other documents at any time.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

> *A Unicode Technical Report (UTR) contains informative material. Conformance to the Unicode Standard does not imply conformance to any UTR. Other specifications, however, are free to make normative references to a UTR.*

Please submit corrigenda and other comments with the online reporting form [Feedback]. Related information that is useful in understanding this document is found in the References section. For the latest version of the Unicode Standard see [Unicode]. See [Reports] for a list of current Unicode Technical Reports. For more information about versions of the Unicode Standard, see [Versions].

Contents

---

1. Scope

This report provides discussion 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 that provide detailed descriptions for particular character properties. Instead it presents a general overview and typology of character properties and property values.

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.

> *D2b Character semantics:* The semantics of a character are determined by its identity, normative properties, and behavior.

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 *UAX#14 Line Breaking Properties* [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 character behavior depends on external context, such as the type and nature of the document, the language of the text, or the cultural expectation 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] as informative and overridable properties.

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 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.

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 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.

> *D9 Normative property:* A Unicode character property whose values are required for conformance to the standard.
>
> *Normative* means that implementations that claim conformance to a particular version of the Unicode Standard 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 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. As a result, *some* of the normative properties disallow any kind of overriding by higher-level protocols. Other normative properties are overridable by higher-level protocols, because their intent is to provide a common basis for behavior, but they may require tailoring for particular local cultural conventions or particular implementations. [examples ommitted and emphasis added]

By making a property normative, the Unicode Standard guarantees that conformant implementations can rely on the fact that other conformant implementations 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. Examples are the bidirectional properties and their use by the bidirectional algorithm [Bidi].

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.

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

2.5 Informative Properties

D9a Informative property: A Unicode character property whose values are provided for information only.

A conformant implementation is free to use or change such values as it may require, while 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 particular properties are being used in distinct ways. 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 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 relying on the specified behavior less interesting to conforming implementations.
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. Assigning a normative property would imply an unwarranted restriction on existing and established practice.

2.6 Referring to Properties

The Property Aliases [Alias] and Property Value Aliases [ValueAlias] define a set of names and abbreviations 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. The names may be translated in appropriate environments, and additional aliases may be useful. The case distinctions, whitespace, and '_' in the property names are not normative and unless a specific form is required in a particular application, all forms are equivalent.

Note: Currently there is at most one abbreviated name and one long name for each property. However, in the future additional aliases may be added. 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 (cc). The combination of property value and property name is, however, unique. For more information, see UTR #18: Regular Expression Guidelines [RegEx].

[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 3.2.0

Then cite that version of the standard, using the standard citation format that is provided for each version of the Unicode Standard. For Unicode 3.2.0, it is:

> The Unicode Consortium. The Unicode Standard, Version 3.2.0, defined by: The Unicode Standard, Version 3.0 (Reading, MA, Addison-Wesley, 2000. ISBN 0-201-61633-5), as amended by the Unicode Standard Annex #27: Unicode 3.1 (http://www.unicode.org/reports/tr27/) and the Unicode Standard Annex #28: Unicode 3.2 (http://www.unicode.org/reports/ tr28/)

2.7 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 documentation files that explain the organization of the database and the format and meaning of the property data in the files. The Unicode Character Database file explains the overall organization of the current version of the UCD and tells which files explain specific data files.

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 position, 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 that case, the original number of the definition is given at the end in square brackets.

Properties and property values

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 [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.

Types of Property Values

PD4. Default Property Value

For a given Unicode property, the 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. [D11]

Note: There may be more than one default value per property.

PD5. 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.

PD6. 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.

PD7. 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.

PD8. Numeric Property

A numeric property can take on any integer, or real value.

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

PD9. Catalog property

A property that is an accumulation of values, unrelated to an algorithm, that, in principle, grows with every version of the Unicode Standard.

Examples are *age* and *block* properties. Both get additional values each time a new version of the Standard is issued that adds new characters or blocks.

Conformance Status of Properties

PD10. Normative Property

A Unicode character property whose values are required for conformance to the standard. [D9]

Note: A normative process that depends in a normative and testable way on a property, 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 - but is is recommended that processes use carefully chosen default values for characters that they do not handle.

PD11. Informative Property

A Unicode character property whose values are provided for information only. [D9a]

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.

PD12. Provisional Property

A Unicode character property whose values are unapproved and tentative, and which may be incomplete or otherwise not in a usable state. [D9b]

Classification of Properties

PD13. Context-independent 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 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.

PD16. Stable Property

A property is stable with respect to a particular algorithm or process, if changes in the assignment of property values are restricted to transformations that are stable with respect to that 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 [Normal].

PD17. Immutable Property

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

An immutable property is trivially stable with respect to all algorithms. Example of immutable, or fixed, properties are the code position and name of each Unicode character.

PD18. Overridable Property

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

See Section 4.2.

PD19. 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 for compatibility.

PD20. Simple property

A Unicode character 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. [D9c]

PD21. Derived Property

A property whose values are algorithmically derived from some combination of simple properties. [D9d]

Other Definitions

PD22. Property alias

A unique identifier for a particular Unicode character property. [D10]

PD23. Property value alias
A unique identifier for a particular enumerated value for a particular Unicode character property. [D10a]

PD24. Higher-level protocol
Any agreement on the interpretation of Unicode characters that extends beyond the scope of this standard. [D8]

String functions

PD25. Offset
An offset into a Unicode string is a number from 0 to *n* where *n* is the length of the string in code units, and 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 offset *n* indicates the position after the last code unit in the string.

PD26. Code point aligned offset
An offset into a Unicode string that is aligned to a code point boundary.

PD27. Substring
A substring S[a,b] is the string formed by all code units of S after offset a and before offset b.

Example: if S is "xyz" then S[1,2] is "y".

PD28. String Function
A string function is a function whose input is a substring.

PD29. Text boundary function
A string function whose value is only defined on substrings of length 0.

Text boundary functions, such as IsBreak(S[a,a]), typically have Boolean values, but a function like LineBreakType(S[b,b]) could return an enumeration.

PD30. Context-independent string function
Given a string X, and substring S = X[a,b], a context-independent string function is any string function F for which F(X[a,b,]) = F(S[0,b-a]) for all X, a and b.

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

PD31. Context dependent string function
Given a string X and substring S = S[a,b], a context-dependent string function is any string function F for which F(X[a,b]) != F(S[0,b-a]) for any X, a or b.

In other words, the input to a context-dependent string function requires information about the code points surrounding the substring as well as the code points in the substring. Any text boundary function is

PD32. Folding Function
A folding function is idempotent context-independent string function.

Idempotent means that the output of the function is a string, and repeated application of the same function produce the same output: F(F(S)) = F(S) for all S.

Every folding establishes a set of equivalence classes that partitions all strings, where X = Y if and only if F(X) = F(Y).

Normalization is an example of a folding.

PD 33. Count preserving string function

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

PD 34. Length preserving string function

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

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.

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 [Unicode] 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 the bidirectional class EN should generally be converted to class AN (for details see UAX#9 Bidirectional Algorithm [Bidi].

• 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.

5. Updating Properties and Extending the Standard

5.1 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

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

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, and need to be changed as information becomes available.

In other cases, there may have been unintentional mistakes in the original information that require corrections. 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+0061 "A" could not be changed to "B"; the general category for U+0061 "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 Stability Policy* [Stability]. Note that the status of a property as normative does not imply a stability guarantee.

Stability of Assignment

Stability of assignment is the definition of an immutable property. For example, once a character is encoded, its code position and name are immutable properties. The main benefit of an immutable property is to allow software and documents to refer to its values without the need to track future updates to the Standard. One side effect of an immutable property is that errata cannot be fixed. For example, mistakes in naming are noted in the nameslist in a note or by using an alias, but the formal name remains unchanged, even in cases of clear-cut typographical errors.

Because the code position is an immutable property, if a character is ever found to not be needed, or to be a mistaken duplicate of an existing character, it will not be removed. Instead, it will be given an additional property, *deprecated*, and its use will be strongly discouraged. However, its identity remains intact, and all existing documents containing the character remain well-defined.

Stability of Result when Applying the Property

Stability of result is the definition 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 past or future versions of Unicode.

However, unlike immutable properties, stable properties may be corrected in exceptional circumstances outlined in [Stability]. For example, the correction must be of an obvious mistake, such as a typographical error, and any alternative would violate the stability of the identity of the character in question. While this makes the stability guarantee less absolute, allowing such exceptions prevents the need for encoding duplicate characters simply to correct clerical or other clear-cut errors in property assignments.

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.

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.

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.

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.

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 better represented via other properties, or it may not make sense to extend the property to new characters. Such 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 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.

6.2 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 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 unassigned code location a implementation defined 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.

6.3 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 might be given a value of 'undetermined', or 'unknown at time of publication'.

Currently no property has been given such values and the conditions under which they would be applied, or in which form, have not yet been defined.

6.4 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.

References

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

[Bidi] The Unicode Consortium. *Unicode Standard Annex #9: The 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] *UAX # 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] *UAX # 14, Line Breaking Properties*
http://www.unicode.org/reports/tr14/

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

[RegEx] Unicode Technical Report #18: *Regular Expression Guidelines* 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 Stability Policy*
http://www.unicode.org/standard/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

[UCD - Doc] Unicode Character Database.
http://www.unicode.org/Public/UNIDATA/UCD.html
For documentation of the contents of the Unicode Character Database and its associated files

[Unicode] The Unicode Consortium. *The Unicode Standard, Version 4.0*. (Reading, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1), or online as http://www.unicode.org/versions/Unicode-4.0.0

[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.

Acknowledgements

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

Revisions

Changes from previous revisions

3 Added several defintions: stable transforms, string functions, catalog, etc.

2 Fixed Summary, Scope and Stability sections, revised and reordered the definitions, updated the Status and References sections, renumbered sections, reworded and fixed typos throughout.

1 First version for public review
