Technical Reports

L2/09-129

Proposed Update Unicode Standard Annex #24

UNICODE SCRIPT PROPERTY

Version	Unicode 5.2.0 draft 1
Authors	Mark Davis (markdavis@google.com), Ken Whistler (ken@unicode.org)
Date	2009-03-02
This Version	http://www.unicode.org/reports/tr24/tr24-12.html
Previous Version	http://www.unicode.org/reports/tr24/tr24-11.html
Latest Version	http://www.unicode.org/reports/tr24/tr24
Revision	12

Summary

This annex specifies an assignment of script ~~names~~ property values to all Unicode code points. This information is useful in mechanisms such as regular expressions and other text processing tasks.

Status

*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time.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

> ***A Unicode Standard Annex (UAX)** forms an integral part of the Unicode Standard, but is published online as a separate document. The Unicode Standard may require conformance to normative content in a Unicode Standard Annex, if so specified in the Conformance chapter of that version of the Unicode Standard. The version number of a UAX document corresponds to the version of the Unicode Standard of which it forms a part.*

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

Contents

1 Introduction

> *Script*: A collection of symbols used to represent textual information in one or more writing systems.

The majority of characters encoded in the Unicode Standard [Unicode] are elements of collections called scripts. Exceptions include symbols, punctuation characters intended for use with multiple scripts, and characters that do not have a stand-alone script identity because they are intended to be used in combination with another character.

Therefore, a text in a given script is likely to consist of characters from that script, together with shared punctuation and characters whose script identity depends on the characters with which they are used.

1.1 Classification of Text by Script ~~Name~~ Property

The Unicode Character Database [UCD] provides a mapping from Unicode characters to script ~~name~~ property values. This information is useful for a variety of tasks that need to analyze a piece of text and determine what parts of it are in which script. Examples include regular expressions or assigning different fonts to parts of a plain text stream based on the prevailing script.

These processes are similar to the task of bibliographers in cataloging documents by their script. However, bibliographers often ignore small inclusions of other scripts in the form of quoted material in cataloging. Conversely, significant differences in the writing style for the same script may be reflected in the bibliographical classification—for example, Fraktur or Gaelic for the Latin script.

Script information is also taken into consideration in collation. The data in the Default Unicode Collation Element Table (DUCET) are grouped by script, so that letters of different script ~~value~~s have different primary sort weights. However, numbers, symbols,

and punctuation are not grouped with the letters. For the purposes of ordering, therefore, script is most significant for the letters. For more information, see Unicode Technical Standard #10, "Unicode Collation Algorithm" [UCA].

These examples demonstrate that the definition of *script* depends on the intended purposes of the classification. *Table 1* summarizes some of the purposes for which text elements can be classified by script.

Table 1. Classification of Text by Script Name

Granularity	Classification	Purpose	Special Values
Document	Bibliographical	Record in which script a text is printed or published; subdivides some scripts—for example, Latin into normal, Fraktur, and Gaelic styles	**Unknown**
Character	Graphological/ typographical	Describe to which script a character belongs based on its origin	
	Orthographical	Describe with which script (or scripts) a character is used	**Common, Inherited**
	For collation	Group letters by script in collation element table	
Run	For font binding or search	Determine extent of run of like script in (potentially) mixed-script text	

Bibliographical, graphological, or historical classifications of scripts need different distinctions than the type of text-processing–related needs supported by Unicode script property values. The requirements of the task not only affect how fine-grained the classification is, but also what kinds of special values are needed to make the system work. For example, when bibliographers are unable to determine the script of a document, they may classify it using a special value for **Unknown**. In text processing, the identities of all characters are normally known, but some characters may be shared across scripts or attached to any character, thus requiring special values for **Common** and **Inherited**.

Despite these differences, the vast majority of Unicode script property values correspond more or less directly to the script identifiers used by bibliographers and others. Unicode script property values are therefore mapped to their equivalents in the registry of script ~~identifiers~~ codes defined by [ISO15924].

1.2 Scripts and Blocks

Unicode characters are also divided into non-overlapping ranges called blocks [Blocks]. Many of these blocks have the same name as one of the scripts because characters of that script are primarily encoded in that block. However, blocks and scripts differ in the following ways:

- Blocks are simply ranges, and often contain code points that are unassigned.
- Characters from the same script may be in several different blocks.
- Characters from different scripts may be in the same block.

As a result, for mechanisms such as regular expressions, using script property values produces more meaningful results than simple matches based on block names.

For more information, see *Annex A, Character Blocks*, in Unicode Technical Standard #18, "Unicode Regular Expressions" [RegEx].

2 Usage Model

The script property values form a full partition of the codespace: every code point is assigned a single script property value. This value is either the value for a specific script ~~value~~, such as **Cyrillic,** or is one of the following three special values:

- **Inherited**—for characters that may be used with multiple scripts, and that inherit their script from the preceding characters. These include~~s~~ nonspacing marks, enclosing marks, and the zero width joiner/non-joiner characters.
- **Common**—for other characters that may be used with multiple scripts.
- **Unknown**—for unassigned, private-use, noncharacter, and surrogate code points.

As new scripts are added to the standard, more script property values will be added. See *Section 3.2, Assignment of Script Property Values.*

The following paragraph occasioned extensive argumentation, and "may need work" still.

~~A character is assigned a specific Unicode script~~ If a character is only regularly used with a single script, then it is given that specific script property value (as opposed to **Common** or **Inherited**) ~~only when it is clearly not used with other scripts~~. This facilitates the use of the script property for common tasks such as regular expressions, but means that some characters that are definite members of a given script ~~by their graphology~~, based on their forms and history, nevertheless are assigned one of the generic values. As more data on the usage of individual characters is collected, the script property value assigned to a character may change. Rarely would a character change from one specific script to another. However, if it becomes established that a character is regularly used with more than one script, it will be assigned the **Common** or **Inherited** script property value~~, where previously it would have had a more specific script value~~. Similarly, if it becomes established that a character is regularly used with only a single, specific script, it will be assigned a specific script property value. ~~However, the opposite type of change is possible as well.~~

The occasional use of character from one script in the context of another script, as for instance the citation of a Greek letter used as a mathematical constant in the midst of Latin text, or the use of a Latin letter in the midst of Han text, is not considered sufficient evidence of "regular use" requiring a designation of **Common** script property value. It is also possible for a character, once given a **Common** or **Inherited** script property value, upon further research, to be changed to a specific script, instead.

2.1 Handling Characters with the Common Script Property

In determining the boundaries of a run of text in a given script, programs must resolve

any of the special script property values, such as **Common,** based on the context of the surrounding characters. A simple heuristic uses the script of the preceding character, which works well in many cases. However, this may not always produce optimal results. For example, in the text "... gamma (γ) is ...", this heuristic would cause matching parentheses to be in different scripts.

Generally, paired punctuation, such as brackets or quotation marks, belongs to the enclosing or outer level of the text and should therefore match the script of the enclosing text. In addition, opening and closing elements of a pair resolve to the same script property
values, where possible. The use of quotation marks is language dependent; therefore it is not possible to tell from the character code alone whether a particular quotation mark is used as an opening or closing punctuation. For more information, see *Section 6.2, General Punctuation,* of [Unicode].

Some characters that are normally used as paired punctuation may also be used singly. An example is U+2019 RIGHT SINGLE QUOTATION MARK, which is also used as *apostrophe,* in which case it no longer acts as an enclosing punctuation. An example from physics would be <ψ| or |ψ>, where the enclosing punctuation characters may not form consistent pairs.

2.2 Handling Combining Marks

Implementations that determine the boundaries between characters of given scripts should never break between a combining mark (a character with General_Category value of Mc, Mn or Me) and its base character. Thus, for boundary determinations and similar sorts of processing, a combining mark—whatever its script property
value—should inherit the script property value of its base character. Spacing combining marks are typically only used with one script and have the corresponding script property value.

The nonspacing marks normally have the **Inherited** script property value to reflect the fact that their script property
value depends on the base character. However, in cases where the best interpretation of a nonspacing mark *in isolation*
would be a specific script, its script property value may be different from **Inherited**. For example, the Hebrew marks and accents are used only with Hebrew characters and are therefore assigned the **Hebrew** script property value.

The recommended implementation strategy is to treat all the characters of a combining character sequence, including spacing combining marks, as having the script property value of the first character in the sequence. This strategy can also be applied to implementations that use extended grapheme clusters; the differences between combining character sequences and extended grapheme clusters are not material for script resolution. For example, rendering generally works best if an entire combining character sequence can be treated as a segment having a single script, using one set of orthographic rules, and ideally using a single font for display. Because of this recommended strategy, even if a combining mark is really only used with a single script, it makes little difference in practice whether the mark has that particular script property value or **Inherited**.

In cases where the first (base) character itself has the **Common** script property value, and it is followed by one or more combining marks with a specific script property value,

such as the Hebrew marks, it may be even better for processing to let the base acquire the script property value from the first mark. This would be the case, for example, if using a graphic symbol as a base to illustrate the placement of nonspacing marks in a particular script. This approach can be generalized by treating all the characters of a combining character sequence (or extended grapheme cluster) as having the script property value of the first non-**Inherited**, non-**Common** character in the sequence if there is one, and otherwise treating all the characters as having the **Common** script property value.

Note that exceptional fallback for rendering may be required for defective combining character sequences or in some cases where a base character and a combining mark have different specific script property values. For example, there may simply be no felicitous way to display a Devanagari combining vowel on a Mongolian consonant base.

2.3 Using Script ~~Names~~ Property Values in Regular Expressions

The script property is useful in regular expression syntax for easy specification of spans of text that consist of a single script or mixture of scripts. In general, regular expressions should use specific script property values only in conjunction with both **Common** and **Inherited**. For example, to distinguish a sequence of characters appropriate for ~~**Greek**~~ Greek text, one would use

```
((Greek | Common) (Inherited | Me | Mn)?)*
```

The preceding expression matches all characters that ~~are either in~~ have a script property value of **Greek** or ~~in~~ **Common** and which are optionally followed by characters ~~in~~ with a script property value of **Inherited**. For completeness, the regular expression also allows any nonspacing or enclosing mark.

Some languages commonly use multiple scripts, so ~~for **Japanese**~~ to distinguish a sequence of characters appropriate for Japanese text one might use

```
((Hiragana | Katakana | Han | Latin | Common) (Inherited | Me | Mn)?)*
```

Note that while it is necessary to include ~~Latin~~ **Latin** in the preceding expression to ensure that it can cover the typical script use found in many Japanese texts, doing so would make it difficult to isolate a run of Japanese inside an English document, for example. For more information, see Unicode Technical Standard #18, “Unicode Regular Expressions” [RegEx].

2.4 Use of the Script Property in Rendering Systems

In rendering systems, it is generally necessary to respect a certain set of orthographic and typographic rules, which vary across the world. For example, the placement of some diacritics which are nominally rendered above their base may be adjusted to be slightly on the side, as is normally the case for Greek. Another example of variation in rendering is the treatment of spaces in justification. In the absence of an explicit specification of those rules, the script property value of the characters involved provides a good first approximation. Typically, a rendering system will partition a text string into segments of homogeneous script (after resolution of the **Common** and **Inherited** occurrences along the lines described in the previous sections), and then apply the rules appropriate to the script of each segment.

2.5 Limitations

The script property values form a full partition of the Unicode codespace, but that partition does not exhaust the possibilities for useful and relevant script-like subsets of Unicode characters.

For example, a user might wish to define a regular expression to span typical mathematical expressions, but the subset of Unicode characters used in mathematics does not correspond to any particular script. Instead, it requires use of the **Math** property, other character properties, and particular subsets of Latin, Greek, and Cyrillic letters. For information on other character properties, see the [UCD].

In texts of an academic, scientific, or engineering nature, the use of isolated Greek characters is common—for example, Ω for ohm; α, β, and γ for types of radioactive decays or in names of chemical compounds; π for 3.1415..., and so on. It is generally undesirable to treat such usage the same as ordinary text in the Greek script. Some commonly used characters, such as μ, already exist twice in the Unicode Standard, but with different script property values.

2.6 Spoofing

The script property values may also be useful in providing users feedback to signal possible spoofing, where visually similar characters (*confusable characters*) are substituted in an attempt to mislead a user. For example, a domain name such as `macchiato.com` could be spoofed with `macchiato.com` (using U+03BF GREEK LETTER SMALL LETTER OMICRON for the first “o”) or `macchiato.com` (using U+0441 CYRILLIC SMALL LETTER ES for the first two “c”s). The user can be alerted to odd cases by displaying mixed scripts with different colors, highlighting, or boundary marks: `macchiato.com` or `macchiato.com`, for example.

Possible spoofing is not limited to mixtures of scripts. Even in ASCII, there are confusable characters such as 0 and O, or 1 and l. For a more complete approach, the use of script property values needs to be augmented with other information such as General_Category values and lists of individual characters that are not distinguished by other Unicode properties. For additional information, see Unicode Technical Report #36, “Unicode Security Considerations” [Security].

3 Values

Table 2 illustrates some of the script property values used in the data file. The short name for the Unicode script property value matches the ISO 15924 code. Further subdivisions of scripts by ISO 15924 into varieties are shown in parentheses. For a complete list of values and short names, see the Property Value Aliases PropertyValueAliases.txt [PropValue]. As with all property value aliases, the script property values in the file are not case sensitive, and the presence of hyphen or underscore is optional. The order in which the scripts are listed here or in the data file is not significant.

Table 2. Unicode Script Property Values and ISO 15924 Codes

Script Property Value	ISO 15924
Common	Zyyy

Inherited	~~Qaai~~ Zinh
Unknown	Zzzz
Latin	Latn (Latf, Latg)
Cyrillic	Cyrl (Cyrs)
Armenian	Armn
Hebrew	Hebr
Arabic	Arab
Syriac	Syrc (Syrj, Syrn, Syre)
Braille	Brai
...	...

Although Braille is not a script in the same sense as Latin or Greek, it is given a script property value in [Data24]. This is useful for various applications for which these script property values are intended, such as matching spans of similar characters in regular expressions.

3.1 Relation to ISO 15924 Codes

ISO 15924: *Code for the Representation of Names of Scripts* [ISO15924] provides an enumeration of four-letter script codes. In the [UCD] file [PropValue], corresponding codes from [ISO15924] are provided as short names for the scripts.

In some cases the match between these script property values and the ISO 15924 codes is not precise, because the goals are somewhat different. ISO 15924 is aimed primarily at the bibliographic identification of scripts; consequently, it occasionally identifies varieties of scripts that may be useful for book cataloging, but that are not considered distinct scripts in the Unicode Standard. For example, ISO 15924 has separate script codes for the Fraktur and Gaelic varieties of the Latin script.

Where there are no corresponding ISO 15924 codes, ~~the~~ private-use ~~ones~~ codes starting with the letter Q are used. Such values are likely to change in the future. In such a case, the Q-names will be retained as aliases in the file [PropValue] for backward compatibility. For example, the older script property value Qaai was retained as an alias for **Inherited**, when the newly defined script code Zinh was added to ISO 15924 and used as the preferred short name for **Inherited** in Unicode 5.2.

3.2 Assignment of Script Property Values

New characters and scripts are continually added to the Unicode Standard. The following ~~methodology is used to assign~~ principle determines the assignment of script property values ~~when new characters are~~ for existing characters and for characters that are newly added to the Unicode Standard:

A. If a character is only regularly used in ~~only~~ one script, ~~assign it to~~ it takes the script property value for that script
B. Otherwise, nonspacing marks (Mn, Me) and zero width joiner/non-joiner are **Inherited**
C. Otherwise, use **Common**

Script values are not immutable. As more data on the usage of individual characters is collected, script values may be reassigned using the above methodology.

3.3 ~~New~~ Script ~~Names~~ Designators in Character and Block Names

Many character names contain a script designator as their first element(s). For example:

- **LATIN** SMALL LETTER S
- **KATAKANA** LETTER SA
- **NEW TAI LUE** LETTER LOW SA
- **PHAGS-PA** LETTER SA

~~The following methodology is used to create names for new scripts added to the Unicode Standard. Script names~~
Character names are guaranteed to be unique even when ignoring case differences and the presence of SPACE or HYPHEN-MINUS. Underscores are not used in character names. In practice, this means that script designators are also unique, and, because they are a part of character names, they are limited to the same characters used in character names:

- Latin letters A–Z ~~or a–z~~
- Digits 0–9
- SPACE and medial HYPHEN-MINUS

Digits do not actually occur in script designators used in character names.

~~Script names are guaranteed to be unique, even when ignoring case differences and the presence of SPACE or HYPHEN-MINUS. Underscores are not used when assigning script names.~~
Many block names, for example, "Latin-1 Supplement", also contain script designators. These script designators are closely (but not precisely) aligned with the script designators used for character names in the corresponding blocks. Similar restrictions apply to script designators as part of block names, except that there is no restriction on the case of letters.

3.4 Script Property Value Aliases

In addition to short names derived from ISO 15924 script codes, as discussed in *Section 3.1, Relation to ISO 15924 Codes*, each script property value is also given a long name as a script property value alias. These long names are also listed in the [UCD] file [PropValue]. They are constructed to be appropriate for use as identifiers. The long or short property value aliases are the identifiers that should be used in regular expressions and similar usages.

Except for the special script property values such as **Common** and **Inherited**, the long name aliases usually correspond to the script designators, with the replacement of SPACE or HYPHEN-MINUS
by underscores, and titlecasing each subpart of the resulting identifier, for consistency with the conventions used for aliases for other Unicode character properties. For example:

- **Latin**
- **Katakana**

- **New_Tai_Lue**
- **Phags_Pa**

As for all property aliases, script property value aliases are guaranteed to be unique within their respective namespace. See the Character Encoding Stability Policies [Stability] for details. When comparing script property value aliases, loose matching criteria which ignore case differences and the presence of spaces, hyphens, and underscores, should be used. See Section 5.7, "Matching Rules", in [UAX44] for explanation of loose matching criteria.

3.5 Script Names

The term *script name* is no longer used as part of the formal specification of the Unicode script property because it tends to be used informally in several ambiguous senses:

1. To designate the orthographic name of a script in the Unicode Standard. For example: **chirilică**, **Кириллица**, or **キリル文字** for **Cyrillic** (Cyrl). Even in English, such names may occasionally include characters not allowed in script designators or script property values. For example: **Hanunóo** or **N'Ko**
2. To designate any variety of writing, some of which may have ISO 15924 script variety codes, such as the **Gaelic** script, and some of which may not, such as the **Hebrew Cursive** script.
3. As a synonym of the term *script designator* as it appears in character or block names. For example: **HANUNOO** or **NKO**
4. As a synonym of the long name alternate of *script property value aliases*. For example: **Hanunoo** (as opposed to the script code **Hano**) or **Nko** (as opposed to the script code **Nkoo**)

Because of these ambiguities, in Unicode contexts where precision of denotation is required, use of the terms *script property value* or *script designator*, whichever may be appropriate, is preferred.

4 Data File

The Scripts.txt data file is available at [Data24]. The format of the file is similar to that of Blocks.txt [Blocks]. The fields are separated by semicolons. The first field contains either a single code point or the first and last code points in a range separated by “..”. The second field provides the script property value for that range. The comment (after a #) indicates the General_Category and the character name. For each range, it gives the character count in square brackets and uses the names for the first and last characters in the range. For example:

```
0B01;       Oriya # Mn ORIYA SIGN CANDRABINDU
0B02..0B03; Oriya # Mc [2] ORIYA SIGN ANUSVARA..ORIYA SIGN VISARGA
```

The value **Unknown** is the default value, given to all code points that are not explicitly mentioned in the data file.

4.1 Script Anomalies for East Asian Symbols

There are a number of compatibility symbols derived from East Asian character sets which have the script property value **Common** but whose compatibility decompositions contain characters with other script property values. In particular, the parenthesized ideographs, circled ideographs, Japanese era name symbols, and Chinese telegraph symbols in the 3200..33FF range contain Han ideographs, and the squared Latin abbreviation symbols in the same range contain Latin (and occasional Greek) letters. Some of these characters have different scripts in their compatibility decompositions. ~~What~~ This means ~~is~~
that script extents calculated on the basis of the script property value of the symbols themselves will differ from script extents calculated on NFKD normalized text, in which these characters decompose into sequences including the Han and/or Latin characters.

The UTC has determined that ~~since~~ because these symbols may be used with multiple scripts in Chinese, Japanese, and/or Korean contexts, their script property value should simply be left as **Common**. There are other, more reliable clues about the behavior of these compatibility symbols, such as their association with East Asian character sets, which can be used by rendering systems to assure their appropriate display and appropriate font choice. This determination is somewhat different from that for the more script-specific parenthesized and circled Hangul and Katakana symbols in the same range, which *are* given specific script property values. At this point keeping the script property
value stable for these compatibility symbols is more useful for implementers than attempting to reconcile this distinction in treatment by modifying ~~script~~ values for them. Implementations that wish to have script property values that are preserved over compatibility equivalence would tailor the script property values for these characters.

Acknowledgments

Thanks to Julie Allen for comments on this annex, including earlier versions. Asmus Freytag added significant sections to the text for Revisions 7 and 9 and assisted in the rewrite of Section 3 for Revision 13. Eric Muller added ~~the new~~ Section 2.4 for Revision 10~~0~~1 and suggested modifications for Section 2.2.

References

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

Modifications

The following summarizes modifications from the previous revision of this annex.

Revision 12

- **Proposed Update** for Unicode 5.2.0
- Made extensive editorial corrections, particularly for the term of art, "script property value". [KW]
- Updated short alias for Inherited from Qaai to Zinh. [KW]
- Rewrote Section 3. Added a new subsection 3.4, to clarify the distinction between script designators and script property value aliases, their respective matching rules, and the use of underscores. Added a new subsection 3.5 to clarify ambiguity in the term script name. [KW]

Revision 11

- Prepared for Unicode 5.1.0 release and updated title. [KW]
- Added surrogates to list of code points which get **Unknown** script value. [KW]
- Added new Section 2.4 regarding use of the script property in rendering systems. [EM]
- Added clarification in Section 2.2 regarding script inheritance in combining character sequences. [MD, EM, KW]
- Added new Section 4.1 noting script anomalies for some East Asian compatibility symbols. [KW]

Revision 10 being a proposed update, only changes between revisions 11 and 9 are noted here.

Revision 9

- Prepared for Unicode 5.0.0 release [AF].
- Added **Unknown**, and made it default value instead of **Common** [AF].

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

Revision 7

- Prepared for Unicode 4.1 release [AF].
- Split section 3.2 and added section 3.3 [AF].
- Major rewrite of Introduction and usage model. [AF].
- Added section on Maintenance and table of classifications types [AF].

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

Revision 5

- Changed to Unicode Standard Annex.
- Added note on the stability of Q names
- Abbreviated the list of values, so that people would not get the mistaken impression that it was complete
- Added note on Braille
- Added note on Mn, Me characters
- Added note on use of scripts with regard to spoofing
- Minor edits

Revision 4

- Updated references, including reference to Property Value Aliases
- Clarified that the list is for illustration only; the definitive values are in the UCD
- Minor edits

Revision 3

- Minor link editing only
