Technical Reports

L2/15-041

Proposed Update

Unicode Standard Annex #24

UNICODE SCRIPT PROPERTY

Version	Unicode 8.0.0 (draft 2)
Editors	Mark Davis (markdavis@google.com), Ken Whistler (ken@unicode.org)
Date	2014-11-24
This Version	http://www.unicode.org/reports/tr24/tr24-23.html
Previous Version	http://www.unicode.org/reports/tr24/tr24-22.html
Latest Version	http://www.unicode.org/reports/tr24/tr24
Latest Proposed Update	http://www.unicode.org/reports/tr24/proposed.html
Revision	23

Summary

This annex describes two related Unicode code point properties. Both properties share the use of Script property values. The Script property itself assigns single script values to all Unicode code points, identifying a primary script association, where possible. The Script_Extensions property assigns sets of Script property values, providing more detail for cases where characters are commonly used with multiple scripts. This information is useful in mechanisms such as regular expressions and other text processing tasks, as explained in the description of the usage model for these properties.

Status

This is a ***draft*** *document which may be updated, replaced, or superseded by other documents at any time.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

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

Standard of which it forms a part.

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

Contents

1 Introduction

This annex defines two properties, Script and Script_Extensions, which can be used to describe the relation of individual Unicode characters to their use within the context of one or more scripts, in the case of ordinary, common writing.

The glossary definition of "script", in the sense used in the Unicode Standard [Unicode], is as follows:

> *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 belong to scripts. Exceptions include symbols and punctuation characters intended for use with arbitrary scripts, and characters intended to be used in combination with other characters.

Text in a given script consists of characters from that script, together with shared punctuation, symbols, and dependent characters whose script identity depends on the characters with which they are used.

1.1 Classification of Text by Script Property

The Unicode Character Database [UCD] provides a mapping from Unicode characters to Script 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) is grouped by script, so that letters of different scripts 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” [UTS10].

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

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 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" [UTS18].

1.3 Script Extensions

Many characters are shared among several scripts, but without the generic usage normally implied by simply assigning the script property value of **Common** to the character. *Table 2* gives examples of such characters. U+30FC is shared across the Hiragana and Katakana scripts, but is not used in other scripts. U+0640 ARABIC TATWEEL is used in Mandaic, Syriac, Manichaean, and Psalter Pahlavi, as well as the Arabic script.

Table 2. Script_Extensions (scx) Examples

Code Point	Scx Property Value	Character Name
30FC	Hira Kana	KATAKANA-HIRAGANA PROLONGED SOUND MARK
0640	Arab Mand Mani Phlp Syrc	ARABIC TATWEEL

Where a character is commonly used in the context of several scripts, and where the list of those scripts can be explicitly enumerated, such usage can be documented with the Script_Extensions property. For details, see the discussion in *Section 2.9* and *Section 3.3*.

1.4 Usage Not Reflected in the Script Property

Many characters are regularly used out of their normal contexts for specialized purposes—for example, for pedagogical use or as part of mathematical, scientific, or scholarly notations. Such uses are not reflected in the assignment of values for either the Script or Script_Extensions properties, because those properties aim rather to reflect ordinary and common usage of characters with a script (or set of scripts). Implementations are cautioned that such "out-of-context" usage of characters does exist and needs to be supported where required, regardless of the Script and Script_Extensions property values for a given character.

2 Usage Model

2.1 Special vs. Explicit Script Property Values

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, 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 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.

All other Script property values are referred to as *explicit* script values, because they each refer to one specific script.

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

If a character is regularly used only with a single script, it is given that specific Script property value (as opposed to **Common** or **Inherited**). This facilitates the use of the script property for common tasks such as regular expressions, but it also means that some characters that are definite members of a given script, 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. 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. 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.

The **Common** Script property value only indicates that a character is used with multiple scripts, but supplies no information about which particular scripts those are. For many applications such a coarse classification may be insufficient; they require further detailed information. For example, a character picker application which organizes characters into visual buckets by script may need to show a **Common** script character in two or more buckets, depending on which particular scripts use that character. Such supplementary classification will depend on the particular usage and is not provided as a normative or informative property in the Unicode Character Database. See *Section 2.8, Multiple Script Values*.

2.2 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.

2.3 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. See *Section 2.8, Multiple Script Values*.

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.

2.4 Using Script 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 text, one would use

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

The preceding expression matches all characters that have a Script property value of **Greek** or **Common** and which are optionally followed by characters 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 example, 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** 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” [UTS18].

The assignment of a Script property value, and in particular of a Script_Extensions property value, is not guaranteed to be stable. The most recently published values always represent the best information available at the time of publication. It is important not to use the Script or Script_Extensions properties in regular expressions if the goal is to match a reproducible, fixed set of characters across versions of the Unicode Standard.

2.5 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.

2.6 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 [UCD].

In texts of an academic, scientific, or engineering nature, Greek characters are frequently used in isolation—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.

2.7 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 `macchiatο.com` (using U+03BF GREEK LETTER SMALL LETTER OMICRON for the first “o”) or `maссhiato.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” [UTR36].

2.8 Multiple Script Values

If a character is regularly used only with a single script, it is given that specific Script property value; otherwise, the Script property value is either **Common** or **Inherited**. These property values do not indicate *which* scripts a character is used with, only that the character is used with more than one script. For example, U+0660 (٠) ARABIC-INDIC DIGIT ZERO is used both with Arabic and with Syriac; similarly, U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK is shared between Hiragana and Katakana. Neither character is typically used with other scripts, such as Latin or Greek.

More precise information about the use of a character with multiple scripts is important for a number of different kinds of processing. The following examples illustrate such cases:

Example 1. Mixed script detection for spoofing.

> Using the Unicode script property alone, for example, will not detect that neither U+0660 (٠) ARABIC-INDIC DIGIT ZERO nor U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK should be mixed with Latin. See [UTS39] and [UTS46].

Example 2. Determination of script runs for text layout.

> The **Common** characters listed in Example 1 should not continue a Latin script run, but instead should only continue runs of certain scripts.

Example 3. Regex property testing.

> For many common tasks, the regex expression [:script=Arab:] is too narrow, because it does not include U+0660 (٠) ARABIC-INDIC DIGIT ZERO, but the expression [[:script=Arab:][:script=Common:]] is far too broad, because it also includes thousands of symbols, plus the U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK. A regex engine can instead specify a regular expression like [:scx=Arab:], which matches based on both the Script property value and the extended script data, and which would include characters such as U+0660 (٠) ARABIC-INDIC DIGIT ZERO. For more information, see Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18].

2.9 Script_Extensions Property

Many of the characters that are assigned a Script property value of **Common** or **Inherited** are not commonly used with *all* scripts, but rather only with a limited set of scripts. The Script_Extensions property is designed to support tasks, such as those outlined above, where it is desirable to know more precisely in which script context such characters can be expected to occur in common use.

Unlike the Script property, the Script_Extensions property consists of a set of values for each character. The Script_Extensions property is primarily targeted at customary modern use of characters, and does not encompass technical usage such as UPA or math. Its values are based on the best available knowledge of usage, which may change over time. The values can be expected to change more frequently than many other Unicode character properties, as more information is gleaned about the usage of given characters. Thus, implementers should be prepared for enhancements and corrections to the values whenever they upgrade to a new version of the property.

The vast majority of characters in the standard are used with only a single script. For those characters, the Script_Extensions property value is a set containing as its single member the Script property value for that character.

Occasionally, even characters that have a Script property value of **Common** or **Inherited** might have a Script_Extensions property value containing only a single script. This does not mean that those characters are used soley with a single script—rather, such characters are known or strongly suspected of being used with multiple scripts. However, reliable information is lacking regarding which other scripts belong in this set. The Script_Extensions property for such characters will be updated in future versions of the standard, if the missing information becomes available.

Conversely, characters for which the Script_Extensions property value contains multiple Script property values typically have a Script property value of either **Common** or **Inherited**. However, in some cases, a character belonging to a particular script may be borrowed for use with one or more other scripts. While the Script property value for such a borrowed character would be the same as the script it is primarily used with, the Script_Extensions property value at times will also include additional scripts. As a result, there is no guarantee that it will always be true that:

$$\mathtt{Script\_Extensions(c)} \neq \{\mathtt{Script(c)}\} \rightarrow (\mathtt{Script(c)} = \mathbf{Common}) \vee (\mathtt{Script(c)} = \mathbf{Inherited})$$

The Script_Extensions property values are given in the file ScriptExtensions.txt in the

Unicode Character Database [UCD].

3 Values

Table 3 illustrates some of the Script property values used in the Scripts.txt 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 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 3. Unicode Script Property Values and ISO 15924 Codes

Script Property Value	ISO 15924
Common	Zyyy
Inherited	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, private-use 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 principle determines the assignment of Script property values for existing characters and for characters that are newly added to the Unicode Standard:

A. If a character is only regularly used in one script, it takes the Script property value for that script
B. Otherwise, ~~use **Common**~~ if the predominant use of the character is in one script, but it is also used in others, then it takes the Script property value associated with that predominant use
C. Otherwise, nonspacing marks (Mn, Me) and zero width joiner/non-joiner are **Inherited**
D. Otherwise, use **Common**

An example of criterion "B" would be the occasional use of an Arabic character in a related minor-use or historic script. In such a case, the predominant use would still be for Arabic, and the Script property value is determined to be **Arabic**, rather than **Common**. The determination of predominant use in such cases is based in part on an estimation of likely frequency of use. This choice is designed to maximize the usefulness of the Script property value for determination of script runs in text, and so on, without having to branch to more elaborate processing to determine how to handle **Common** property values by examining the Script_Extensions value set in these edge cases. The choice of an explicit Script property value, instead of **Common** or **Inherited**, in these edges cases is done when, in the judgement of the Unicode Technical Committee, that explicit Script property value is a reasonable default.

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 Assignment of Script_Extensions Property Values

The following principle determines the assignment of Script_Extensions property values for existing characters and for characters that are newly added to the Unicode Standard:

A. If a character has the Script property value of **Common** or **Inherited**, and in principle might occur with almost any script, its Script_Extensions is also {**Common**} or {**Inherited**}, respectively
B. If a character is regularly or occasionally used in more than one script, but such usage is limited to a small, enumerable list, then the character takes the Script_Extensions property value consisting of the set of Script property values for each of those scripts
C. Otherwise, the Script_Extensions property value defaults to a set containing a single value, the Script property value for that code point

Examples of characters that have the Script property value of **Common** or **Inherited**, but in principle might occur with almost any script, would include many symbol characters. They simply get a Script_Extensions default value of {**Common**} or

{**Inherited**}. Only when the common usage consists of a relatively small and well-determined list of scripts is it useful to enumerate that set explicitly for a Script_Extensions property value. In many cases such sets may involve shared typographical traditions between neighboring or related scripts. Note that assignment of an enumerated set of more than one Script property values to the Script_Extensions property value for a character can occur both in cases where that character has the Script property value **Common** or **Inherited** and in cases where it has an explicit Script property value such as **Arabic**.

Script_Extensions property values are not immutable. As more data on the usage of individual characters is collected, Script_Extensions property values may be adjusted. This may occur either as a result of the Script property value for the character being changed, or as a result of a determination that a given character is used with more (or fewer) scripts than earlier determined.

3.4 Script 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

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
- Digits 0–9
- SPACE and medial HYPHEN-MINUS

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

3.5 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 5.9, Matching Rules*, in [UAX44] for explanation of loose matching criteria.

3.6 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.

3.7 Script Anomalies

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.

Examples of such characters are listed in *Table 4*. Some of these characters have different scripts in their compatibility decompositions. This means 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.

Table 4. Examples of East Asian Symbols with Common Script Values

U+249C (⒜) PARENTHESIZED LATIN SMALL LETTER A
U+24B6 (Ⓐ) CIRCLED LATIN CAPITAL LETTER A
U+1F130 (🄰) SQUARED LATIN CAPITAL LETTER A
U+3382 (㎂) SQUARE MU A
U+1F12A (🄪) TORTOISE SHELL BRACKETED LATIN CAPITAL LETTER S
U+3192 (㆒) IDEOGRAPHIC ANNOTATION ONE MARK
U+3220 (㈠) PARENTHESIZED IDEOGRAPH ONE
U+3244 (㉄) CIRCLED IDEOGRAPH QUESTION
U+3280 (㊀) CIRCLED IDEOGRAPH ONE
U+32C0 (㋀) IDEOGRAPHIC TELEGRAPH SYMBOL FOR JANUARY
U+3358 (㍘) IDEOGRAPHIC TELEGRAPH SYMBOL FOR HOUR ZERO
U+337B (㍻) SQUARE ERA NAME HEISEI
U+33E0 (㏠) IDEOGRAPHIC TELEGRAPH SYMBOL FOR DAY ONE

The UTC has determined that 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. Examples of such characters are shown in *Table 5*.

Table 5. Examples of East Asian Symbols with Kana or Hangul Script Values

U+32D0 (㋐) CIRCLED KATAKANA A
U+3260 (㉠) CIRCLED HANGUL KIYEOK
U+3200 (㈀) PARENTHESIZED HANGUL KIYEOK
U+3300 (㌀) SQUARE APAATO

There are other symbols not constrained to primary use in East Asian contexts, which have the **Common** script, but where some users would expect to have a specific script. Examples are shown in *Table 6*. Symbols in such cases are assigned to the **Common** script because they may be used with a wide variety of scripts, and are not necessarily limited to the script values of their compatibility decompositions.

Table 6. Examples of Other Symbols with Common Script Values

U+2122 (™) TRADE MARK SIGN
U+2120 (℠) SERVICE MARK
U+00A9 (©) COPYRIGHT SIGN
U+210F (ℏ) PLANCK CONSTANT OVER TWO PI
U+2109 (℉) DEGREE FAHRENHEIT
U+214D (⅍) AKTIESELSKAB

At this point keeping the Script property value stable for these compatibility symbols is more useful for implementers than attempting to reconcile these distinctions in treatment by modifying 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.

4 Data Files

The data files associated with the Unicode script property are available at [Data24].

Scripts.txt

The format of this 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.

ScriptExtensions.txt

The format of this data file is similar to Scripts.txt, except that the second field contains a space-delimited list of short Script property values. That list defines the set of Script property values which constitute the Script_Extension property value for that code point. For example:

```
# Script_Extensions=Arab Syrc

064B..0655    ; Arab Syrc # Mn  [11] ARABIC FATHATAN..ARABIC HAMZA BELOW

# Script_Extensions=Arab Mand Mani Phlp Syrc
```

```
0640          ; Arab Mand Mani Phlp Syrc # Lm       ARABIC TATWEEL
```

The default value for the Script_Extensions property for a code point not explicitly listed in ScriptExtensions.txt is a set containing one value: the Script value of that code point.

Acknowledgments

Mark Davis authored the initial versions. Ken Whistler has added to and maintains the text of this annex.

Thanks to Julie Allen for comments on this annex, including earlier versions. Asmus Freytag added significant sections to the text for Revisions 7, 9 and 19 and assisted in the rewrite of Section 3 for Revision 13. Eric Muller added Section 2.4 (now 2.5) for Revision 11 and suggested modifications for Section 2.3.

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 23 [KW]

- **Proposed Update** for Unicode 8.0.0.
- Clarification added regarding the choice of Script value for a character when its Script_Extensions value set contains more than one value in *Section 3.2 Assignment of Script Property Values*.
- Minor edits.

Revision 22 [KW]

- **Reissued** for Unicode 7.0.0.
- Updated some values for the Script_Extensions property.
- Minor edits.

Revision 21 [KW]

- **Reissued** for Unicode 6.3.0.
- Minor edits.

Revision 20 being a proposed update, only changes between revisions 21 and 19 are noted here.

Revision 19 [KW]

- **Reissued** for Unicode 6.2.0.

- Updated information about Script_Extensions property, to reflect its change from provisional to informational.
- Revised the Summary, to reflect the updated scope.
- Rewrote *Section 1 Introduction*.
- Added new *Section 1.3 Script Extensions*.
- Added new *Section 1.4 Usage Not Reflected in the Script Property*.
- Added new header for *Section 2.9 Script_Extensions Property* and rewrote content for that section.
- Added new *Section 3.3, Assignment of Script_Extensions Property Values* and adjusted numbering of other sections.
- Added disclaimer about stability of Script and Script_Extensions property values in *Section 2.4, Using Script Property Values in Regular Expressions*.
- Applied consistent capitalization to the phrase "Script property value", to match capitalization conventions used for other property names in the standard.

Revision 18 being a proposed update, only changes between revisions 19 and 17 are noted here.

Revision 17 [KW, MD]

- **Reissued** for Unicode 6.1.0.
- Updated text explaining ScriptExtensions.txt, to account for the change of status from a provisional data file to a data file defining a new provisional property, Script_Extensions.
- Moved section 4.1 to be 3.6 Script Anomalies, broadened name, added more cases.

Revision 16 being a proposed update, only changes between revisions 17 and 15 are noted here.

Revision 15

- **Reissued** for Unicode 6.0.0.
- Minor editorial updates. [KW]
- Added *Section 2.8, Multiple Script Values* and new cross-references to it. [MD]
- Changed *Section 4 Data Files* to add discussion of ScriptExtensions.txt. [MD]

Revision 14 being a proposed update, only changes between revisions 15 and 13 are noted here.

Revision 13

- **Reissued** for Unicode 5.2.0
- Made extensive editorial corrections, particularly for the term of art, "script property value". [KW]
- Added paragraph in Section 2 explaining that the Common script value does not indicate what scripts a Common script character is actually used with. [KW]

- Added the term "explicit script value" to Section 2, Usage Model, and added a header to what is now subsection 2.1 to clarify the structure of the section. [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 12 being a proposed update, only changes between revisions 13 and 11 are noted here.

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

Copyright © 1999–2014 Unicode, Inc. All Rights Reserved. The Unicode Consortium makes no expressed or implied warranty of any kind, and assumes no liability for errors or omissions. No liability is assumed for incidental and consequential damages in connection with or arising out of the use of the information or programs contained or accompanying this technical report. The Unicode Terms of Use apply.

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