Technical Reports

L2/20-023

Proposed Update Unicode® Standard Annex #24

UNICODE SCRIPT PROPERTY

Version	Unicode 13.0.0 (draft 2)
Editors	Ken Whistler (ken@unicode.org)
Date	2019-10-17
This Version	http://www.unicode.org/reports/tr24/tr24-30.html
Previous Version	http://www.unicode.org/reports/tr24/tr24-29.html
Latest Version	http://www.unicode.org/reports/tr24/tr24
Latest Proposed Update	http://www.unicode.org/reports/tr24/proposed.html
Revision	30

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*This is a **draft** document which may be updated, replaced, or superseded by other documents at any time.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

The concept of *script* is a key organizational principle for the Unicode Standard [Unicode]. This annex introduces the general concept of script and the specific ways in which the concept is used in the standard. Two character properties, Script and Script_Extensions, are then specified in detail.

A *script* is a collection of letters and other written signs that generally has the following attributes:

- The written elements share a common graphological style and history.

- The collection is used (in full, or as a subset) to represent textual information in a writing system for one or more languages.

For example, the Russian language is written with a distinctive set of letters, as well as other marks or symbols that together form a subset of the *Cyrillic* script. Other languages using the Cyrillic script, such as Ukrainian or Serbian, employ a different subset of those letters.

Normally, the letter shapes of one script are unrelated to those of another script. So, for example, the letter shapes of the Cyrillic script share nothing in common with the letter shapes of the Hebrew script. However, writing systems may be historically related to each other, in which case there are often systematic *similarities* in letter shapes and occasional identical shapes. So because the Cyrillic script is historically related to the Greek script, those two scripts share a significant number of letter forms.

A script may also explicitly borrow letters from another script. For example, some writing systems that use the Cyrillic script have borrowed letter forms from the Latin script. Furthermore, letter forms may show accidental similarity in shapes: a simple line or circle used as a letter, for example, could have been independently created many times in the history of the development of writing systems.

The writing system for a language occasionally employs more than one script. The best known example is the Japanese language, whose writing system uses four scripts: the Han ideographs (*kanji*), as well as the Hiragana and Katakana syllabaries, but also a subset of the Latin letters.

Some languages may have competing writing systems that use different scripts, or change scripts from one historical period to another. For example, the Turkish language was historically written in the Arabic script but is now written using the Latin script. For many other languages there are similar cases, where an historical writing system used one script, while a modern writing system for the same language may use a different script.

Some scripts, such as the Latin script or the Arabic script, have an historically developed cosmopolitan status, and are used for the representation of the writing systems of hundreds or even thousands of different languages. The *script* in such cases consists of the complete set of letters and other signs needed to represent *all* of the writing systems covered, which may include historical as well as modern text forms, rather than simply being a single alphabet or other set of graphic symbols needed for writing a single language.

1.1 Examples of Script Classification

Independent of its use by the Unicode Standard, there are distinct needs for classification by script. For example, writing systems can be classified by the script or scripts they use. In cases of continuous historical derivation of scripts from predecessor scripts, an existing graphological classification may consider a writing system to be using a variant of an ancestor script, whereas the Unicode Standard may give each historic stage its own script identity for the purposes of character encoding.

In another example, bibliographers need to catalog documents by the primary script in which they are written. In so doing, bibliographers often ignore small inclusions of other scripts in the form of quoted material, for the purpose of catalog identification. Conversely, significant differences in writing style for the same script may be reflected in the bibliographical classification—for example, Fraktur or Gaelic styles for the Latin script.

Such stylistic distinctions are ignored in the Unicode Standard, which treats them as presentation styles of the Latin script.

Bibliographers also assign a single classification code for Japanese or Korean documents, even though the respective writing systems use a mix of scripts. Such single codes have also proven useful as a shorthand notation for describing the repertoires of characters needed when supporting identifiers, as for the Internationalized Domain Names (IDN).

1.2 Script Identity and Unicode

The Unicode Standard fundamentally considers characters as elements of scripts in making encoding decisions. For example, when a letter is borrowed from one script into another, it often is encoded again as a distinct element of the borrowing script. This occurs most often in the case for letters. For punctuation and other similar marks, the decision may instead be made to explicitly designate a character for common use with all scripts, or to document its use with a defined subset of all scripts.

In addition to letters, the Unicode Standard includes many graphic symbols which fall outside the scope of particular writing systems and are not associated with particular scripts. For example, there are commonly used punctuation marks such as commas and quotation marks that are widely shared across scripts. The same consideration applies to the European digits "1", "2", "3", ... The Unicode Standard also contains many combining marks intended to be used in multiple writing systems, as well as symbols for notational systems like mathematics that have their own rules and identity independent of writing systems for particular languages.

1.3 Scripts and Blocks

Unicode characters are divided into non-overlapping ranges called blocks [Blocks]. Many of these blocks have a name derived from a script name, 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 encoded in several different blocks.
- Characters from different scripts may be encoded in the same block.

As a result, using the block names as simplistic substitute for script identity generally leads to poor results. For example, see *Annex A, Character Blocks*, in Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18].

1.4 Script Classification in Text Processing

In text processing the classification of text by script is by necessity more fine-grained than when cataloging documents. The classification by script is essential 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. For all of these tasks, the challenge is to break a text into script runs, or stretches of text that are all treated as belonging to the same script.

Script information is also taken into consideration in collation, so that strings are grouped by script when sorted. To that end, the Default Unicode Collation Element Table (DUCET) assigns letters of different scripts different ranges of primary sort weights. However, numbers, symbols, and punctuation are not grouped with the letters. For the purposes of

ordering, therefore, explicit script identity is most significant for the letters. For more information, see Unicode Technical Standard #10, “Unicode Collation Algorithm” [UTS10].

These examples demonstrate that the use of *script* (and to a certain extent, its exact specification) 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	

1.5 Classification of Text by Script Property

The exact way in which one uses script information about text depends on the kind of processing that is involved. In addition to being normally less-fine-grained, bibliographical, graphological, or historical classifications of scripts need different distinctions than common text processing-related tasks. To assist in the development of interoperable implementations for text processing that depends on script classification, the Unicode Standard defines two character properties, Script and Script_Extensions.

The Script property assigns a single value to each character, either explicitly associating it with a particular script, or assigning one of several special values. The Script property is discussed in detail in Section 2, *The Script Property*. The Script_Extensions property builds on this model, by better documenting cases where characters are neither used solely with members of a single script nor shared universally. The Script_Extensions property is unusual in that each of its values is a set of Script values. The Script_Extensions property is discussed in detail in Section 3, *The Script_Extensions Property*.

The special property values required to support text-processing needs are different from those needed in other classifications. 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 in focus, the vast majority of Unicode Script property values correspond more or less directly to the script identifiers used by bibliographers and others.

This annex documents the definition and use of those properties and describes the data files in the Unicode Character Database [UCD] that specify exact values of those properties for all Unicode characters.

1.6 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). Implementers 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 The Script Property

2.1 Script Property Values

The Script property is an enumerated property of type *catalog*. Its values form a full partition of the codespace: every Unicode code point is assigned a single Script property value. This value is either the explicit 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 a preceding base character. These include nonspacing combining marks and enclosing combining marks, as well as U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER.
- **Common**—for other characters that may be used with multiple scripts.
- **Unknown**—for unassigned, private-use, noncharacter, and surrogate code points.

Collectively, these three special values are called *implicit* values, in contrast to all other Script property values, which each refer to one specific script and which are called *explicit* values.

As new scripts are added to the standard, explicit Script property values will be added to the enumeration. Implementations are advised to allow for this growth in enumerated values. See also Section 2.3, *Initial Assignment of Script Property Values*.

The implicit values **Common** or **Inherited** do not indicate *which* scripts a character is used with—only that the character is used with more than one script. For example, U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK is shared between Hiragana and Katakana and is not typically used with other scripts, such as Latin or Greek. 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. For data on

which scripts a character is commonly used with, see Section 3, *The Script_Extensions Property*.

A value of **Inherited** means that the character is treated as if it had the Script property value of a preceding base character. (See Section 5.2, *Handling Combining Marks*.) Where the character is not part of a combining sequence, as is the case for U+200C ZERO WIDTH NON-JOINER and U+200D ZERO WIDTH JOINER, there are special script inheritance rules for use in text run processing.

The Script property values assigned for all characters are specified in the file Scripts.txt [Data24] in the Unicode Character Database [UCD]. A complete enumeration of Script property values and their short names in provided in [PropValue]. For further discussion, see Section 4, *Data Files*.

2.2 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 [PropValue], where feasible, the short name for the Unicode Script property value matches the corresponding ISO 15924 code, as exemplified in *Table 3*.

Table 3. Unicode Script Property Values and ISO 15924 Codes

Script Property		ISO 15924
Long	Short	
Common	Zyyy	Zyyy
Inherited	Zinh, Qaai	Zinh
Unknown	Zzzz	Zzzz
Latin	Latn	Latn (Latf, Latg)
Cyrillic	Cyrl	Cyrl (Cyrs)
Coptic	Copt, Qaac	Copt
Armenian	Armn	Armn
Georgian	Geor	Geor (Geok)
Hebrew	Hebr	Hebr
Arabic	Arab	Arab (Aran)
Syriac	Syrc	Syrc (Syrj, Syrn, Syre)
Braille	Brai	Brai
Han	Hani	Hani (Hans, Hant)
...	...	...

In some cases the match between the 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. Such codes for script varieties are shown in parentheses in *Table 3*.

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 then used as the preferred short name for **Inherited** starting in Unicode 5.2.

2.3 Initial 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, 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, for regular expressions, 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. However, some characters that are definitely members of a given script, based on their forms and history, nevertheless are assigned one of the implicit Script values instead.

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

Script values are not immutable. 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 an explicit 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.

2.4 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.

2.5 Script Property Value Aliases

In addition to short names derived from ISO 15924 script codes, as discussed in *Section 2.2, 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 [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 implicit Script property values **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.

2.6 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.

2.7 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 Script Value = Common

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 (1月) IDEOGRAPHIC TELEGRAPH SYMBOL FOR JANUARY
U+3358 (0点) IDEOGRAPHIC TELEGRAPH SYMBOL FOR HOUR ZERO
U+337B (平成) SQUARE ERA NAME HEISEI
U+33E0 (1日) 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 Katakana 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 Script Value = Common

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.

3 The Script_Extensions Property

Where a character is commonly used in the context of several scripts, it is often desirable to know more precisely in which script context such characters can be expected to occur. The implicit Script property values **Common** and **Inherited** were originally designed simply to indicate that a character, such as a punctuation mark, occurs widely in conjunction with many scripts, rather than being associated with use for just one script. However, many of the characters that are assigned a value of **Common** or **Inherited** are not commonly used with *all* scripts, but rather only with a limited set of scripts. In cases where the list of such scripts can be explicitly enumerated, it can help various processing to have the list specified. Such lists of use by a character across several scripts are documented with the Script_Extensions (scx) property.

The Script_Extensions property is implemented as sets of Script property values, known as *scx sets* ("Es Cee Ex sets"). *Table 7* gives examples of scx sets for various Unicode code points, along with their Script and General_Category property values. Note that for completeness, default values for scx sets are given for all Unicode code points, including reserved code points and noncharacters. The details of assignment of scx set values are discussed further below.

Table 7. Script_Extensions Examples

Code	Scx Set	Script	Gc	Character Name
Scx set contains one implicit Script value				
0020	{Common}	Common	Zs	SPACE
0301	{Inherited}	Inherited	Mn	COMBINING ACUTE ACCENT
243F	{Unknown}	Unknown	Cn	<reserved-243F>
FFFF	{Unknown}	Unknown	Cn	<noncharacter-FFFF>
Scx set contains one explicit Script value				
0061	{Latn}	Latin	Ll	LATIN SMALL LETTER A
0363	{Latn}	Inherited	Mn	COMBINING LATIN SMALL LETTER A
1CD1	{Deva}	Inherited	Mn	VEDIC TONE SHARA
Scx set contains multiple explicit Script values; Script(cp) is implicit				
30FC	{Hira Kana}	Common	Lm	KATAKANA-HIRAGANA PROLONGED SOUND MARK
3099	{Hira Kana}	Inherited	Mn	COMBINING KATAKANA-HIRAGANA VOICED SOUND MARK

1CD0	{Beng Deva Gran Knda}	Inherited	Mn	VEDIC TONE KARSHANA
1802	{Mong Phag}	Common	Po	MONGOLIAN COMMA
060C	{Arab Rohg Syrc Thaa Yezi}	Common	Po	ARABIC COMMA
0640	{Adlm Arab Mand Mani Phlp Rohg Sogd Syrc}	Common	Lm	ARABIC TATWEEL
Scx set contains multiple explicit Script values; Script(cp) is explicit				
096F	{**Deva** Dogr Kthi Mahj}	Devanagari	Nd	DEVANAGARI DIGIT NINE
09EF	{**Beng** Cakm Sylo}	Bengali	Nd	BENGALI DIGIT NINE
1049	{Cakm **Mymr** Tale}	Myanmar	Nd	MYANMAR DIGIT NINE

For example, U+30FC KATAKANA-HIRAGANA PROLONGED SOUND MARK is shared across the Hiragana and Katakana scripts, but is not used in other scripts, so it is assigned an scx set value of {Hira Kana}. U+0640 ARABIC TATWEEL is used in Adlam, Mandaic, Manichaean, Psalter Pahlavi, Hanifi Rohingya, Sogdian, and Syriac, as well as the Arabic script, but is not used with non-cursive scripts or with scripts unrelated to that family of writing systems, so it is assigned an scx set value of {Adlm Arab Mand Mani Phlp Rohg Sogd Syrc}.

The Script_Extensions property is primarily targeted at customary modern use of characters, and does not encompass technical usage such as phonetic transcriptional systems or mathematics.

3.1 Script_Extensions Property Values

This section describes formal construction and constraints on the Script_Extensions (scx) property values.

A. Each code point is associated with exactly one non-empty set of values of the sc property. This set is known as the code point's *scx set*.

Unlike most other character properties, all values of the scx property constitute sets of values. The empty set is not allowed; the scx value for unassigned, private use, and non-character code points is the set { **Unknown** }.

B. The elements of the scx set consist of an unordered list of unique values of the Script (sc) property values.

The scx values { **Latn Grek** } and { **Grek Latn** } are identical; for ease of comparison, the values in the sets may be sorted and listed in alphabetical order.

C. An scx set either contains a single implicit sc value or one or more explicit sc values.

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.

D. If the sc property value of a code point is explicit, then that value must be an element of the scx set for that code point as well.

Even though there is no formal constraint on the number of explicit values that may occur in an scx set, it is unlikely that any scx value would individually list even a majority of existing scripts. The implicit sc value **Common** is intended instead for use in those cases where a character is in very widespread use across many scripts.

There are no formal rules specifying when a particular sc value must be added to the scx set for a particular assigned character. Whether to document that a character is used with multiple scripts via the Script_Extensions property remains a judgment call, and is always based on the best information available to the Unicode Technical Committee.

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 solely 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. Examples illustrating this can be seen in *Table 7*, where the Samavedic tone mark U+1CD0 VEDIC TONE KARSHANA is attested at least for Devanagari, Bengali, Kannada, and Grantha, but where U+1CD1 VEDIC TONE SHARA is only known (for now) to occur in Devanagari Samavedic texts. The Script_Extensions property for such characters will be updated in future versions of the standard, if better 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. Examples can be seen in *Table 7* for shared sets of digits. It is common for one Indic script to use digits from another script; Devanagari digits are known, for example, to also be used in Dogra, Kaithi and Mahajani. As a result of this kind of borrowing across scripts, there is no guarantee that it will always be true that:

```
Script_Extensions(c) ≠ {Script(c)} → (Script(c) = Common) ∨ (Script(c) = Inherited)
```

Table 8 provides examples of scx sets that are not allowed, according to the well-formedness rules for scx sets.

Table 8. Examples of Disallowed (Ill-formed) Scx Sets

Scx Set	Script	Problem Description
{Latn}	Unknown	Set contains an explicit value for Script(cp)=Unknown
{Common}	Inherited	Set contains an implicit value that does not match Script(cp)
{Latn Latn}	Latn	Same value occurs more than once in the set
{Inherited	Inherited	More than one implicit value occurs in the set

Common}		
{Latn Common}	Latn	Explicit and implicit values both occur in the set
{Latn Grek}	Hani	Script(cp) does not occur in the list of explicit values

The complete list of Script_Extensions scx set values are specified in the file ScriptExtensions.txt in the Unicode Character Database [UCD].

3.2 Initial 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 value is {**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. 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.

4 Data Files

The data files associated with the Unicode Script property are available in the Unicode Character Database. See [Data24].

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

4.2 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=Adlm Arab Mand Mani Phlp Rohg Sogd Syrc

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

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

4.3 PropertyValueAliases.txt

This file provides the complete enumerated list of all Script property values: both long and short names. As for all property value aliases, the Script property values listed in the PropertyValueAliases.txt are not case sensitive, and the presence of hyphen or underscore is optional. The aliases are listed alphabetically, but that order is only a convenience for reference and is not otherwise significant. See [PropValue].

5 Implementation Notes

This section discusses various topics related to the implementation of the Script property and the Script_Extensions property.

5.1 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.

5.2 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. 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.

5.3 Multiple Script Values

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 Script property alone, for example, will not detect that the U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK (Script=**Common**) should not be mixed with Latin. See [UTS39] and [UTS46].

Example 2. Determination of script runs for text layout.

> U+30FC (ー) KATAKANA-HIRAGANA PROLONGED SOUND MARK 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+060C ARABIC COMMA, 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 the Script_Extensions property value, and which would include *appropriate* Script=**Common** characters such as U+060C ARABIC COMMA. For more information, see Unicode Technical Standard #18, "Unicode Regular Expressions" [UTS18].

5.4 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 might 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.

5.5 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.

5.6 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.

5.7 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 `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” [UTR36].

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, 19, and 26 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 30 [KW]

- **Proposed Update** for Unicode 13.0.0.
- Added Yezi to scx set listed for 060C.
- Small textual edits.

Modifications for previous versions are listed in those respective versions.
