L2/17-234

Define and Stabilize Value Format of Unicode Version

Behnam Esfahbod – Virgule Typeworks – July 20, 2017
Individual Contribution – For consideration by UTC

Introduction

This document is a proposal to expand the description of Unicode Version with definition of its value format and stabilize this definition. The main goal is to help implementations with having a stable API for the Unicode Version, as well as other values related to Unicode Version, such as the Unicode Age character property.

1. Background

1.1. Unicode Version

The *About Versions of the Unicode® Standard* page defines Unicode Version as:

> *Version numbers for the Unicode Standard consist of three fields, denoting the major version, the minor version, and the update version, respectively. For example, "Unicode 3.1.1" indicates major version 3 of the Unicode Standard, minor version 1 of Unicode 3, and update version 1 of minor version Unicode 3.1.* [Versions]

From this description, it is not clear that the three fields of Unicode Version—major, minor and update—are guaranteed to be integer values.

And, with the assumption that these fields are integers, no details are provided to help with implementation, and technically, these numbers may be larger than any common type in common programming languages. Some implementations assume the value fits in a 8-bit integer (signed or unsigned), some go with 32-bit integers, and some other even larger.

1.2. Age Property

The *Unicode Standard Annex #44: Unicode Character Database* defines Character Age as:

> *The Age property indicates the first version in which a particular Unicode character was assigned. For example, U+20AC € EURO SIGN was added to Version 2.1 of the Unicode Standard, so it has age=2.1, while U+20B9 ₹ INDIAN RUPEE SIGN was added to Version 6.0 of the Unicode Standard, so it has age=6.0.*
>
> *The short values for the Age property for assigned (designated) code points are of the form "m.n", with the first field corresponding to the major version, and the second field corresponding to the minor version. There is no need for a third version field, because new characters are never assigned in update versions of the standard. The long values for the Age property for assigned code points start with a "V" and use an underscore instead of a dot between the major and minor version numbers: V2_1, V6_0, and so on. This makes the long format more useful as an identifier in programming languages. It is also useful in regular expressions, where the dot has other significance.*
>
> *The default value of the Age property, used for unassigned (undesignated) code points, is expressed with labels that depart from the numerical versioning scheme of the Age property for assigned code points; the short form for the default is "NA", and the long form for the default is "Unassigned". Implementations of parsers which manipulate the Age property need to be prepared for this special case, rather than expecting the default value to be expressed numerically, as "0.0", for example.*
>
> *The Age property is based on when a character is encoded in the standard. It is normative and immutable, and cannot be meaningfully tailored.* [UAX44]

Basically, this definition relies on the definition of Unicode Version on the details of the version fields. Again, without clear value format, building a stable API to return numerical values for this property is impossible.

1.3. Derived Age Property Data File

The data file for Derived Age Property does not provide any details about the value format, either. Basically, it relies on the definition of the *short values* from UAX #44 for the values. [DerivedAge]

2. Current Implementations

Looking at how implementations handle Unicode Version, we can see that there's big inconsistency between libraries, even in the same project. Some implementations make assumptions about the value format and limits for the numerical fields, while some other use string values, mainly to stay flexible in case of future changes to the value format.

2.1. Programming Languages

Traditionally, programming languages do not have API to expose the Unicode Version of their internal Unicode data, nor API for the Age character property.

For the new programming languages with such APIs, majority of them rely on ICU library for these. PHP/Hack (HHVM), Java, and Swift are examples of these languages.

2.2. Libraries

Since it's common practice to rely on third-party libraries for access to these Unicode data, even for programming languages, we look at the API of some widely-used libraries, specially ICU.

ICU4C

The C/C++ implementation of ICU (`icu4c`) uses an unsigned 8-bit integer type (`uint8_t`) for fields of Unicode Version, as implemented in its `UVersionInfo` int-array type. [ICU Versions]

ICU4J

The Java implementation of ICU (`icu4j`) uses Java's signed 32-bit integer type (`int`) for fields of Unicode Version, as implemented in its `VersionInfo` class. [ICU Versions]

HarfBuzz

In its *UCDN - Unicode Database and Normalization*, HarfBuzz uses a C string type for Unicode Versions, stored internally as `UNIDATA_VERSION` string literal.

3. Value Format

Based on the common understanding of Unicode Version, the details needed for implementations appear to be as follows. The proposal is to add these format details to the definition of Unicode Versions. [Versions]

Field Type

Based on current common interpretation of the aforementioned definition, it is assumed to be an *integer*.

The proposal is to explicitly define the fields being integer numbers, which cannot include any non-numeric parts.

Minimum Value

It is commonly understood to be one (1) for the *major* field, and zero (0) for the *minor* and *update* fields.

The proposal is to explicitly define a minimum value of zero (0) for all components.

Maximum Value

A maximum value is not mentioned anywhere in the standard and there is no consensus on how large the value can get. Trying to go with the smallest type commonly used in practice may not be a wise decision in this case, because that would be 127, which is fairly short and may reach easily in a few years or decades if the Unicode release schedule changes in the future.

Therefore, the proposal is to define a maximum value for the fields that can pass a few hundreds. Largest value fitting in a signed 16-bit integer, 32767, could be a good choice.

3.1. Proposed text for Value Format

> **Version Numbering**
>
> Version numbers for the Unicode Standard consist of three fields, denoting the major version, the minor version, and the update version, respectively. Each field is a numerical value, no smaller than zero (0) and no larger than <maximum-value>.

For example, “Unicode 3.1.1” indicates major version 3 of the Unicode Standard, minor version 1 of Unicode 3, and update version 1 of minor version Unicode 3.1.

4. Stabilization

For implementations to be able to build a stable API, they need to rely on stability of the value format of Unicode Version. Therefore, the proposal is to add a stability policy to the *Unicode Character Encoding Stability Policies* [Stability] regarding the value format of Unicode Version numbers.

4.1. Proposed text for Stability Policy

Unicode Version Stability

Applicable Version: Unicode [X].0+

The format of Unicode Version values will not be changed. In particular, the Unicode Version for any future update to the Unicode Standard is guaranteed to fit in the same value format.

This policy ensures that implementers can always depend on the type of fields of the version of the Unicode Standard. The Unicode Standard may not use all possible values in this format, specifically, there is no Unicode Version with major field value of zero (0).

Also, the Unicode Standard guarantees that newer version of the standard will always have values larger than any previous version. The ordering of these values is based on natural numeric ordering of their fields.

A. References

[Versions] About Versions of the Unicode Standard

Information on version numbering, and citing and referencing the Unicode Standard, the Unicode Character Database, and Unicode Technical Reports:
http://www.unicode.org/versions/

[UAX44] Unicode Standard Annex #44: Unicode Character Database

Latest version:
http://www.unicode.org/reports/tr44/
Version 10.0.0:
http://www.unicode.org/reports/tr44/tr44-20.html

[DerivedAge] Derived Age Property Data File

Latest version:
http://www.unicode.org/Public/UCD/latest/ucd/DerivedAge.txt
Version 10.0.0:
http://www.unicode.org/Public/10.0.0/ucd/DerivedAge.txt

[Stability] Unicode Character Encoding Stability Policy

http://www.unicode.org/policies/stability_policy.html

[ICU Versions] ICU Version Numbers

http://site.icu-project.org/processes/release/tasks/versions