L2/22-087

Profile Changes in UAX #31 / UTS #39

To: UTC
From: Robin Leroy, Mark Davis, Source code working group
Date: 2022-04-14

While the source code ad hoc working group has not finished their work, there is agreement that additional changes could be made to UAX #31 and UTS #39 to avoid some problems with identifier profiles, and especially those that allow for ZWJ/ZWNJ.

I. Proposed changes to UAX #31

1. In Section 2 Default Identifiers of UAX #31, just before R1a, add:

Beyond such minor modifications, profiles can be used to significantly extend the character set available in identifiers. In so doing, care must be taken not to unintentionally include undesired characters, or to violate important invariants.

A property-based set should only be added in a profile if it corresponds to the intent.

For instance, consider a profile that adds subscript and superscript digits and operators in order to support technical notations (*e.g.*, identifiers such as the Assyriological `dun₃⁺`, the chemical `Ca²⁺_concentration`, the mathematical `xₖ₊₁` or `f⁽⁴⁾`, or the phonetic `daan⁶`). That profile may be described as adding the following set to XID_Continue:

`[⁽⁾⁺₊⁼₌⁻₋⁰₀¹₁²₂³₃⁴₄⁵₅⁶₆⁷₇⁸₈⁹₉]`.

It may seem more principled, instead of listing these characters in a targeted fashion, to include them via properties or combinations of properties that include the desired ones. This is unadvisable, however.

For instance, `\p{General_Category=Other_Number}` is the general category set containing the subscript and superscript digits. But it also includes the compatibility characters `⑴1.`, which do not serve the aforementioned technical notations, and are very likely inappropriate for identifiers—on multiple counts. A language that allows currency symbols in identifiers could have `\p{General_Category=Currency_Symbol}` as a profile, since that property matches the intent.

Similarly, a profile based on adding entire blocks is sure to include unintended characters, or to miss ones that are desired; on the use of blocks see Annex A, *Character Blocks*, in [UTS18].

Defining a profile by use of a property also needs to take account of the fact that unless the property is designed to be stable (such as XID_Continue), code points could be removed in a future version of Unicode. If the profile is also to have stable identifiers (backwards compatible), then measures need to be taken to support that. See *UAX31-R1b. Stable Identifiers.*

When defining a profile, it is also critical to ensure that it is compatible with the normalization chosen for the identifiers. The example of subscripts and superscripts above preserves identifier closure under Normalization Forms C and D, *but not* KC and KD. Under NFKC and NFKD, the subscript and superscript parentheses and operators normalize to their ASCII counterparts. A language using that profile should conform to UAX31-R4 using NFC, not NFKC.

Implementations defining a profile that includes the ZERO-WIDTH JOINER or ZERO-WIDTH NON JOINER characters should implement requirement UAX31-R1a.

2. In the section R1a. Restricted Format Characters of UAX #31, reword the requirement as follows:

***UAX31-R1a*. Restricted Format Characters:** *To meet this requirement, an implementation shall define a profile for UAX31-R1 which allows format characters, but shall restrict their use to the contexts A1, A2, and B defined ~~as described~~ in Section 2.3, Layout and Format Control Characters.*

3. In R1a, just before R1b, add:

Note that the ZWJ and ZWNJ characters in R1a are not in XID_Continue, and that meeting the requirement R1. Default Identifiers does *not* require supporting R1a. Restricted Format Characters (or for that matter, R1b. Stable Identifiers).

These ZWJ and ZWNJ characters are invisible in most contexts, and only added to Default Identifiers in a declared profile. They have security and usability implications that make them inappropriate for implementations that do not carefully consider those implications. For example, they should not be added via a profile where spoofing concerns are paramount, such as top-level domain names.

The purpose for R1a is to describe how to restrict the usage of ZWJ and ZWNJ to reduce the impact, for those implementations that choose to support them.

4. 2.3 Layout and Format Control Characters Make the following change in Paragraph 1:

Note that the ZWJ and ZWNJ characters in R1a are not in XID_Continue, and that meeting the requirement R1. Default Identifiers does *not* require supporting R1a. Restricted Format Characters (or for that matter, R1b. Stable Identifiers).

These ZWJ and ZWNJ characters are invisible in most contexts, and only added to Default Identifiers in a declared profile. They have security and usability implications that make them inappropriate for implementations that do not carefully consider those implications. For example, they should not be added via a profile where spoofing concerns are paramount, such as top-level domain names.

The purpose for R1a is to describe how to restrict the usage of ZWJ and ZWNJ to reduce the impact, for those implementations that choose to support them.

5. Before Section 2.3.1 Limitations, insert the following note:

Note: The restrictions in A1, A2, and B are similar to the CONTEXTJ rules defined in Appendix A, *Contextual Rules Registry*, in *The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)* [IDNA2008].

II. Proposed changes to properties

6. In the property files IdentifierStatus.txt and IdentifierType.txt, remove the Joiner_Control characters ZWJ and ZWNJ from Identifier_Type=Inclusion and Identifier_Status=Allowed.
    - This will result in their reverting to Identifier_Type=Default_Ignorable, and thereby Identifier_Status=Restricted.

III. Proposed changes to UTS #39

7. Below Table 1. Identifier_Status and Identifier_Type of UTS #39, add the following notes:

Note: In Unicode 15.0, the Joiner_Control characters (ZWJ/ZWNJ) have been removed from Identifier_Type=Inclusion. They thereby have the properties Identifier_Type=Default_Ignorable and Identifier_Status=Restricted.

Their inclusion in programming language identifier profiles has usability and security implications. Moving them to Restricted can help to avoid problems for implementations that simply add the characters in Identifier_Status=Allowed to an identifier profile.

This step does not prevent them from being included in an identifier profile as per [UAX31], but lessens the likelihood that they are inadvertently included, without proper consideration.

[Review Note:
Further changes may be made to Identifier_Type=Inclusion in the future, based on ongoing work to provide guidance to implementation on avoiding source code spoofing issues.]

For example, that could result in other changes to Identifier_Type=Inclusion. Most of these characters are less dangerous than the Joiner_Controls, but still should only be used in identifiers by implementations that accept a broader spectrum of characters, and understand the security and usability implications.]

IV. Rationale

On the need for guidance about profiles

There is very little guidance given in UAX #31 on the way in which one should define profiles for default identifiers that substantially alter the identifier space; instead the examples focus on profiles addressing compatibility concerns, and profiles that are minor adjustments such as allowing a leading low line. Many programming languages have however found a need for such extensions.

With the concrete example of subscripts and superscripts (inspired by existing practice, see below), we showcase the fact that properties do not usually make for good profiles.

Another example of disparate set which should generally not be used as a profile by programming languages is the set Identifier_Type=Inclusion; its name should certainly not be taken to mean that it is recommended to use the complete set as a profile in default identifiers: indeed that would be difficult for most programming languages, as that set contains the ASCII Pattern_Syntax characters [-:.'].

However, within that set, ZWJ and ZWNJ still stand out as having specific usability and security implications that the other characters do not.

On ZWJ and ZWNJ in identifiers

As described in Section 2.3 *Layout and Format Control Characters* of UAX #31, these characters have a visible effect in some contexts, and no visible effect in others.

The source code ad hoc working group found that it is a problem

1. if visually equivalent identifiers are logically distinct,

but also

2. if logically equivalent identifiers are visually distinct.

UAX #31 currently recommends that when allowed in identifiers, ZWJ and ZWNJ be ignored when comparing identifiers. Failing to follow that recommendation leads to problems of the first kind. However, even following that recommendation and ignoring them addresses the first kind of problem, but leads to problems of the second kind. Indeed, consider the following program:

```
std::string نامهای;  // `names`.
{ // Narrower scope.
  std::string نامهٔای;  // `a letter`.
  نامهای += "Mark";
}
```

It looks like `"Mark"` is being added to the variable نامهای (names), but if ZWNJ is ignored when comparing identifiers, it is actually being added to the variable نامهٔای (a letter), which unexpectedly shadows نامهای (names)!

These issues, which can have security as well as usability implications, are specific to the ZWJ and ZWNJ, and are not shared by the other Identifier_Type=Inclusion characters. For that reason, the working group recommends that ZWJ and ZWNJ be removed from Identifier_Type=Inclusion and thus from Identifier_Status=Allowed.

On the example of subscripts and superscripts

The definition of XID_Start and XID_Continue is ultimately based on general categories, so that the subscript and superscript digits, being [:No:] rather than [:Nd:], were excluded, whereas subscript and superscript letters, being [:Lm:], and mathematical alphanumerics, being [:Lo:] and [:Ln:], were included.

However, these digits, along with the subscript and superscript operators, are used in a variety of technical notations (homophone indices in Assyriology, atom numbers and ionization states in chemistry, indexing, powers, derivatives, and more in mathematics, tone numbers in various phonetic notations, etc.). Some programming languages have therefore seen fit to include them in their definition of identifiers.

Notably, C++11 through C++20 allows them; specifically its definition of identifiers allows ranges of the *Latin-1 Supplement* which include the characters `[¹²³]`, as well as the range U+2070 through U+218F, which starts with the entirety of the *Superscripts and Subscripts* block. The Julia programming language also allows these, by having Other Numbers as part of its Continue set, and explicitly allowing the subscript and superscript operators and parentheses.

While both the block-based approach and the general category approach end up including far more characters than is desirable (indeed, both languages end up allowing "⑴" in identifiers), the characters in the example profile are in actual use in identifiers in these languages, see, e.g., the following GitHub searches[1]:

— `x₂` (Julia, C++);
— `χ²` (Julia, C++);
— `aᵢ₊₁` (Julia)
— `xₖ₊₁` (C++, Julia);
— `xₙ₊₁` (Julia).

As programming language designers use non-trivial profiles in order to support these usages, we need to provide better guidance on constructing profiles that better match their intent, and on the potential for complications involved, in advance of the more extensive work in development by the working group.

[1] Note that GitHub does not support lexical analysis in searches, so that comments are found as well as identifiers. All searches listed find identifiers. Note also that, since GitHub does not support regular expressions in search, one needs to look for specific expressions, leading to small numbers.