Update Annex E onto Unicode 15.1

Steve Downey <sdowney@gmail.com>
<sdowney2@bloomberg.net>

Document #:	D3727R0
Date:	2025-05-31
Project:	Programming Language C++
Audience:	SG16, CWG

Abstract

Update the non-normative Annex E to reflect Unicode 15.1 in order to close Core Issue [CWG2843].

Contents

1 Introduction

“Unicode Standard Annex #31: Unicode Identifiers and Syntax” for Unicode 15.1 [UAX31-15.1:online] clarifies some of the rules for identifiers and white space syntax. This paper proposes no normative changes in either area. It simply updates the current Annex E to match the forms used in Unicode Annex 31 as of Unicode 15.1 [unicode_15_1]. See also the differences from Unicode 14 as marked up in [UAX31-15.1-DIFF:online]

2 Proposal

Strike the reference to R1a Restricted Format Characters. Note that as of Unicode 15.1, U+200D ZERO WIDTH JOINER and U+200C ZERO WIDTH NON-JOINER are part of XID_Continue, and allowed in identifiers.

Replace individual disclaimed conformance points with blanket disclaimer as the conformance list is open ended.

3 Wording

The proposed changes are relative to [N5008].

Update Annex E as follows.

�.1 General [uaxid.general]

1 This Annex describes the choices made in application of UAX #31 (“Unicode Identifier and Pattern Syntax”) to C++ in terms of the requirements from UAX #31 and how they do or do not apply to this document. In terms of UAX #31, this document conforms by meeting the requirements R1 “Default Identifiers” and R4 “Equivalent Normalized Identifiers” from UAX #31. The other requirements from UAX #31, also listed below, are either alternatives not taken or do not apply to this document.

�.2 R1 Default identifiers [uaxid.def]

�.�.1 General [uaxid.def.general]

1 UAX #31 specifies a default syntax for identifiers based on properties from the Unicode Character Database, UAX #44. The general syntax is

```
<Identifier> := <Start> <Continue>* (<Medial> <Continue>+)*
```

where `<Start>` has the XID_Start property, `<Continue>` has the XID_Continue property, and `<Medial>` is a list of characters permitted between continue characters. For C++ we add the character U+005F LOW LINE, or _, to the set of permitted `<Start>` characters, the `<Medial>` set is empty, and the `<Continue>` characters are unmodified. In the grammar used in UAX #31, this is

```
<Identifier> := <Start> <Continue>*
<Start> := XID_Start + @\textrm{\ucode{005f}}@
<Continue> := <Start> + XID_Continue
```

2 This is described in the C++ grammar in 5.11, where *identifier* is formed from *identifier-start* or *identifier* followed by *identifier-continue*.

�.�.2 R1a Restricted format characters [uaxid.def.rfmt]

1 The clause R1a has been removed from UAX #31.

> The characters that were added when meeting this requirement are now part of the default; the contextual checks required by this requirement remain as part of the General Security Profile in Unicode Technical Standard #39, “Unicode Security Mechanisms”.

�.�.3 R1a Restricted format characters [uaxid.def.rfmt]

1 If an implementation of UAX #31 wishes to allow format characters such as U+200D ZERO WIDTH JOINER or U+200C ZERO WIDTH NON-JOINER it must define a profile allowing them, or describe precisely which combinations are permitted.

2 C++ does not allow format characters in identifiers, so this does not apply.

�.�.4 R1b Stable identifiers [uaxid.def.stable]

1 An implementation of UAX #31 may choose to guarantee that identifiers are stable across versions of the Unicode Standard. Once a string qualifies as an identifier it does so in all future versions.

2 C++ does not make this guarantee, except to the extent that UAX #31 guarantees the stability of the XID_Start and XID_Continue properties.

�.3 R2 Immutable identifiers [uaxid.immutable]

1 An implementation may choose to guarantee that the set of identifiers will never change by fixing the set of code points allowed in identifiers forever.

2 C++ does not choose to make this guarantee. As scripts are added to Unicode, additional characters in those scripts may become available for use in identifiers.

�.4 R3 Pattern_White_Space and Pattern_Syntax characters [uaxid.pattern]

1 UAX #31 describes how formal languages such as computer languages should describe and implement their use of whitespace and syntactically significant characters during the processes of lexing and parsing.

2 This document does not claim conformance with this requirement from UAX #31.

�.5 R4 Equivalent normalized identifiers [uaxid.eqn]

1 UAX #31 requires that implementations describe how identifiers are compared and considered equivalent.

2 This document requires that identifiers be in Normalization Form C and therefore identifiers that compare the same under NFC are equivalent. This is described in 5.11.

�.6 Requirements of UAX #31 for which no claims are made [uaxid.nonobservance]

1 UAX #31 version 15.1 has conformance requirements which either do not apply to C++ or which this document makes no claim about.

(1.1) — R2. Immutable Identifiers

(1.2) — R3. Pattern_White_Space and Pattern_Syntax Characters

(1.3) — R5. Equivalent Case-Insensitive Identifiers

(1.4) — R6. Filtered Normalized Identifiers

(1.5) — R7. Filtered Case-Insensitive Identifiers

(1.6) — R8. Hashtag Identifiers

2 This document also makes no claim about additional conformance points in any versions of UAX #31 in versions after 15.1.

�.7 R5 Equivalent case-insensitive identifiers [uaxid.eqci]

1 This document considers case to be significant in identifier comparison, and does not do any case folding. This requirement from UAX #31 does not apply to this document.

�.8 R6 Filtered normalized identifiers [uaxid.filter]

1 If any characters are excluded from normalization, UAX #31 requires a precise specification of those exclusions.

2 This document does not make any such exclusions.

�.9 R7 Filtered case-insensitive identifiers [uaxid.filterci]

1 C++ identifiers are case sensitive, and therefore this requirement from UAX #31 does not apply.

�.10 R8 Hashtag identifiers [uaxid.hashtag]

1 There are no hashtags in C++, so this requirement from UAX #31 does not apply.

4 Impact on the standard

No normative change. Modifies non-normative text describing the conformance points with the Unicode Identifier standard.

Document history

- **Initial**
  - Rework of P3717R0 on Unicode 15.1

References

[CWG2843] Jonathan Wakely. CWG2843: Undated reference to unicode makes c++ a moving target. `https://wg21.link/cwg2843`, 1 2024.

[N5008] Thomas Köppe. N5008: Working draft, programming languages — c++. `https://wg21.link/n5008`, 3 2025.

[UAX31-15.1-DIFF:online] Mark Davis and Robin Leroy. Unicode identifiers and syntax. `https://www.unicode.org/reports/tr31/tr31-38.html`, 08 2023. (Accessed on 2025-05-31).

[UAX31-15.1:online] Mark Davis and Robin Leroy. Unicode identifiers and syntax. `https://www.unicode.org/reports/tr31/tr31-39.html`, 09 2023. (Accessed on 05/31/2025).

[unicode_15_1] The Unicode Consortium. The unicode standard, version 15.1. `https://www.unicode.org/versions/Unicode15.1.0/`, 2023.