Doc. no.: WG14/N1518
WG21/N3146
PL22.16/10-0136
Date: 2010-10-04
Reply to: Clark Nelson
Phone: +1-503-712-8433
Email: clark.nelson@intel.com

Recommendations for extended identifier characters for C and C++

Introduction

In response to their 2010 FCD ballot, WG21 received the following comment (designated as CA 24) from the Canadian national body:

> A list of issues related TR 10176:2003
>
> 1. "Combining characters should not appear as the first character of an identifier." Reference: ISO/IEC TR 10176:2003 (Annex A) This is not reflected in FCD.
> 2. Restrictions on the first character of an identifier are not observed as recommended in TR 10176:2003. The inclusion of digits (outside of those in the basic character set) under *identifer-nondigit* is implied by FCD.
> 3. It is implied that only the "main listing" from Annex A is included for C++. That is, the list ends with the Special Characters section. This is not made explicit in FCD. Existing practice in C++03 as well as WG 14 (C, as of N1425) and WG 4 (COBOL, as of N4315) is to include a list in a normative Annex.
> 4. Specify width sensitivity as implied by C++03: `\uFF21` is not the same as `A`. Case sensitivity is already stated in [lex.name].

It is reasonable to expect that WG14 would receive a very similar comment in response to an upcoming ballot.

Background

In investigating what various standards say about extended characters in identifiers, the following facts came to light.

General

1. The C++ WD now cites TR 10176:2003 for the specification of extended identifier characters.
2. The C standard (and WD) incorporates the lists from TR 10176:1998 for the specification of valid extended identifier characters.
3. There are differences between the lists in the C standard and in TR 10176:2003.
4. The lists in TR 10176:2003 are recommended as the **minimum** set of characters that an implementation should allow in identifiers.
5. The C standard explicitly allows implementations to accept other characters (implementation-defined) outside the basic source character set in identifiers.
6. The C++ standard (and WD) gives no such permission.
7. In C, a UCN not in the set of those specified as allowed is undefined behavior.
8. In C++, a UCN not in the set of those specified as allowed requires a diagnostic.
9. TR 10176 is based on the same principles as the "Default Identifier Syntax" defined in Unicode UAX#31 (see [DefId]): very roughly, characters defined to be letters are allowed initially, characters defined to be digits and combining marks are allowed non-initially.
10. The identifier character set in XML 1.0 was originally defined using the same principles.
11. For the sake of stability in the presence of an expanding character set, UAX#31 also defines an "Alternative Identifier Syntax" (see [AltId]), in which anything is allowed but: white space, "syntax" characters, private use characters, surrogates,

control characters, and non-characters.
12. Unicode defines "syntax" characters to include most punctuation and symbols (including mathematical operators).
13. XML 1.1 also defines the identifier character set generously, excluding only certain characters and character ranges.
14. The latest edition of XML 1.0 has adopted the identifier specification from XML 1.1.

Combining characters

1. The C standard (and presumably TR 10176:1998) does not list combining characters, compatibility presentation forms, or fullwidth or halfwidth forms as valid in identifiers.
2. TR 10176:2003 has a separate list, also in Annex A, for combining characters, compatibility presentation forms, fullwidth & halfwidth forms, etc.
3. The C++ standard references Annex A, without even acknowledging that it has two parts.
4. TR 10176:2003 recommends that a combining character should not appear as the first character of an identifier.

Digits

1. TR 10176:2003 gives digits as an example of a kind of character often not allowed initially, and calls them out as a separate category in Annex A, but makes no actual recommendation.
2. In C, a UCN representing a digit is not allowed to start an identifier.
3. The C++ standard has no such restriction.

Halfwidth and fullwidth variants

1. TR 10176:2003 makes no recommendation with respect to halfwidth or fullwidth variants, but notes that COBOL (in particular) considers halfwidth and fullwidth variants to be equivalent to the original character (at least in some contexts).
2. The C standard does not explicitly include halfwidth or fullwidth variants as identifier characters.
3. The C++ standard is silent on this topic.

Discussion

General

WG14 and WG21 have experienced that trying to keep a language standard in synch with an expanding character set definition can be problematic. The Unicode Consortium and the World Wide Web Consortium have both acknowledged this, and provided (normative) guidance to avoid the problem. Although TR 10176 is based on problematic principles, the recommendation that **at least** the specified characters should be accepted in identifiers can be completely satisfied using the [AltId] approach.Therefore, it seems reasonable to abandon the [DefId] and TR 10176 approach, in favor of something simpler and more stable.

In a sane world, C and C++ would use the same definition of valid extended identifier characters. In an ideal world, the definition would appear in the C standard, and be referenced by the C++ standard. The publication schedules of WG14 and WG21 would appear to put an ideal world out of reach. But putting textually identical specifications in annexes of both C and C++, based on the [AltId] principles, would seem to be feasible.

Specifics

Estalishing the principle by which the set of valid extended identifier characters will be defined is clearly not enough; it is also necessary to select the exact definition.

In an ideal world, it would be possible to cite a definition from a different standard (as C++ was going to attempt to do by referencing [TR2003]). Citing the identifier syntax from [XML2008] would be problematic, for several reasons:

1. It's not an ISO standard.
2. It covers so much more than just the identifier syntax.
3. Its identifier syntax allows some inappropriate characters (including the basic source characters `"-"`, `"."` and `":"`, and several punctuation marks used in CJK text), and disallows some

inappropriate characters (including "ª", "µ" and "º", which were allowed in C99).

The content and organization of UAX#31 would make it easier to cite. It's still not an ISO standard, but that may not be an insuperable problem. In general, the current definition of [AltId] seems to be better than [XML2008] at tracking recent assignments of blocks in Unicode. But the exact definition of [AltId] has what appears to be a very serious flaw.

[AltId] defines a property called "Pattern_White_Space"; characters having that property are disallowed as identifier characters.Several other categories of characters are also disallowed, including "Pattern_Syntax" characters and control characters. The "Pattern_White_Space" category includes the ASCII SPACE character, several control characters (including HT, LF and CR), and a few others (including the Unicode LINE SEPARATOR and PARAGRAPH SEPARATOR characters) — but it does not include several other characters defined as spaces, including the Latin-1 NO-BREAK SPACE (NBSP).

In addition, it should be noted that [AltId] seems to be unique among standards and recommendations for identifiers, in having no restriction on characters allowed initially.

Combining characters

There is a fairly serious technical reason why an identifier should not start with a combining character: a combining character combines, semantically and visually, with the character **preceding** it. So disallowing this would prevent potentially serious, gratuitous confusion. Thus TR 10176 (reasonably) recommends that it be disallowed.

But [AltId] imposes no restrictions on the first character of an identifier. On the other hand, [XML2008] disallows (some) general combining characters initially, but not any script-specific combining character.

It should be noted that the C and C++ standards published to date appear to scrupulously disallow all combining characters in identifiers — both general and script-specific.Therefore, it has not previously been necessary to place restrictions on whether to allow them initially. It is difficult to imagine how to continue to avoid this problem using an extended identifier character definition based on [AltId].

Digits

While an identifier starting with a script-specific digit might be confused with a number by a human (who recognizes the digit as such), it will not be so confused by a C or C++ compiler — unless the compiler has been specifically extended to recognize script-specific numerical literals. Thus the problem of initial script-specific digits is less severe than the problem of initial combining characters (which is inherent in the structure of Unicode). TR 10176 does not go so far as to actually recommend that they be disallowed — it only mentions that they might be disallowed.

To date, C++ has never allowed script-specific digits as identifier characters, initially or otherwise. C allows them. While C disallows them initially, the "shall" imposing the requirement is not in a "Constraint" section, which of course means that implementations have not, strictly speaking, been required to diagnose them; instead, an identifier starting with a script-specific digit yields undefined behavior.

Halfwidth and fullwidth variants

This is the tip of a very large iceberg; see UAX#15 (if you dare) for much, much more information. Basically, width variations are just one of a dozen or so ways in which one character, or sequence of characters, can be confused for another. Unicode defines four different normalization forms, which can be used to resolve these confusions. The fact that there are multiple normalization forms can reasonably be taken as an indication of the complexity of the situation.

Any standard (including COBOL and TR 10176) that talks about width variations, and no other form of canonical or compatibility equivalence, is very likely demonstrating a potent combination of hubris and ignorance.

I think the C and C++ standards should be silent on this whole topic. An mplementer should be able to decide whether his implementation should normalize or not, and if so which normalization form should be used, based on his understanding of the needs of his customers. The implication of that would be that users should never name different things using identifiers that would normalize to the same string, nor attempt to reference something using anything but its exact name (for example, by using a name that would normalize to the same string as the original name).

Recommendations

The definition of the ranges of UCNs allowed in an identifier should appear in an annex in each standard; the text of these two annexes should be identical. (The wording of the citations of these annexes will be different between the two standards.)

The set of UCNs **disallowed** in identifiers in C and C++ should exactly match the specification in [AltId], **with the following additions**: all characters in the Basic Latin (i.e. ASCII, basic source character) block, and all characters in the Unicode General Category "Separator, space".

General combining characters, appearing in blocks dedicated to that purpose, should be disallowed as the initial character of an identifier. (Script-specific combining characters would be allowed initially, only because of the additional complexity and instability of specifying them.)

There should be no restriction on script-specific digits initally in an identifier.

Proposed wording

The annex specifying the ranges of allowed identifier characters should be Annex D in the C standard, and Annex E in the C++ standard. It should have two sub-clauses: one for allowed identifier characters, and one for characters disallowed initially.

Annex

X. Universal character names for identifier characters (normative)

X.1 Ranges of characters allowed

00A8, 00AA, 00AD, 00AF, 00B2-00B5, 00B7-00BA, 00BC-00BE, 00C0-00D6, 00D8-00F6, 00F8-00FF

0100-167F, 1681-180D, 180F-1FFF

200B-200D, 202A-202E, 203F-2040, 2054, 2060-206F

2070-218F, 2460-24FF, 2776-2793, 2C00-2DFF, 2E80-2FFF

3004-3007, 3021-302F, 3031-303F

3040-D7FF

F900-FD3D, FD40-FDCF, FDF0-FE44, FE47-FFFD

10000-1FFFD, 20000-2FFFD, 30000-3FFFD, 40000-4FFFD, 50000-5FFFD, 60000-6FFFD, 70000-7FFFD, 80000-8FFFD, 90000-9FFFD, A0000-AFFFD, B0000-BFFFD, C0000-CFFFD, D0000-DFFFD, E0000-EFFFD

X.2 Ranges of characters disallowed initially

0300-036F, 1DC0-1DFF, 20D0-20FF, FE20-FE2F

Citing the annex from C

Change 6.2.4.1p3:

> Each universal character name in an identifier shall designate a character whose encoding in ISO/IEC 10646 falls into one of the ranges specified in annex D, subclause D.1.71) The initial character shall not be a universal character name designating a ~~digit~~ character whose encoding falls into one of the ranges specified in subclause D.2. An implementation may allow multibyte characters that are not part of the basic source character set to appear in identifiers; which characters and their correspondence to universal character names is implementation-defined.

Citing the annex from C++

Change 2.11p1:

> An identifier is an arbitrarily long sequence of letters and digits. Each universal-character-name in an identifier shall designate a character whose encoding in ISO 10646 falls into one of the ranges specified in ~~Annex A of TR 10176:2003~~ annex E, subclause E.1. The initial element shall not be a universal-character-name designating a character whose encoding falls into one of the ranges specified in subclause E.2. Upper- and lower-case letters are different. All characters are significant.19

It may also be appropriate to delete the normative reference to TR 10176:2003. Even though the standard will (substantially) follow its recommendations for extended identifier characters, there will remain no actual reference to it.

References

[AltId]
: Unicode Standard Annex #31: Unicode Identifier and Pattern Syntax, "Alternative Identifier Syntax", http://www.unicode.org/reports/tr31/tr31-11.html#Alternative_Identifier_Syntax

[DefId]
: Unicode Standard Annex #31: Unicode Identifier and Pattern

Syntax, "Default Identifier Syntax", http://www.unicode.org/reports/tr31/tr31-11.html#Default_Identifier_Syntax

[TR2003]
ISO/IEC TR 10176:2003 (WG 20 DTR ballot draft), http://www.open-std.org/JTC1/sc22/WG20/docs/n970-tr10176-2002.pdf

[TR2003a]
Online text of identifier character repertoire recommended by [TR2003], http://www.iso.org/ittf/ISOIEC_TR_10176_2003_Table.txt

[XML2006]
Extensible Markup Language (XML) 1.0 (Fourth Edition), "Common Syntactic Constructs", http://www.w3.org/TR/2006/REC-xml-20060816/#sec-common-syn

[XML2008]
Extensible Markup Language (XML) 1.0 (Fifth Edition), "Common Syntactic Constructs", http://www.w3.org/TR/2008/REC-xml-20081126/#sec-common-syn

---

Appendix: [AltId] and [XML2008] illustrated

The Basic Multilingual Plane

Legend: Characters disallowed by [AltId] are indicated by a $\boxed{\text{box}}$. Characters disallowed by [XML2008] are indicated by a gray background.

Every block is presented, most only by name. Certain blocks (especially including those corresponding to the ASCII and Latin-1 "legacy encodings") have significant numbers of punctuation and symbol characters; these are presented character by character. Each assigned, non-control character appears in the HTML source as the appropriate character entity; if it doesn't display correctly in your browser, the fault is almost certainly in your browser setup. The formal name of each character is also present an HTML title, which will hopefully pop up if you hover your mouse pointer over the character.

It should be noted that [AltId] is unique in having no distinction between characters allowed initially and non-initially in an identifier. Where [XML2008] makes such a distinction, it is indicated below as a note. There are also notes for a few isolated (plausible) non-identifier characters.

0000	**Basic Latin**																
0000	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	
0010	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	
0020	☐	$\boxed{!}$	$\boxed{"}$	$\boxed{\#}$	$\boxed{\$}$	$\boxed{\%}$	$\boxed{\&}$	$\boxed{'}$	$\boxed{(}$	$\boxed{)}$	$\boxed{*}$	$\boxed{+}$	$\boxed{,}$	$\boxed{-}$	$\boxed{.}$	$\boxed{/}$	XML disallows $\boxed{-}$ and $\boxed{.}$ initially
0030	0	1	2	3	4	5	6	7	8	9	$\boxed{:}$	$\boxed{;}$	$\boxed{<}$	$\boxed{=}$	$\boxed{>}$	$\boxed{?}$	XML allows $\boxed{:}$ initially, disallows digits initially
0040	$\boxed{@}$	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
0050	P	Q	R	S	T	U	V	W	X	Y	Z	$\boxed{[}$	$\boxed{\backslash}$	$\boxed{]}$	$\boxed{\hat{}}$	_	
0060	$\boxed{`}$	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
0070	p	q	r	s	t	u	v	w	x	y	z	$\boxed{\{}$	$\boxed{\vert}$	$\boxed{\}}$	$\boxed{\sim}$	☐	
0800	**Latin-1 Supplement**																
0080	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	
0090	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	
00A0		$\boxed{¡}$	$\boxed{¢}$	$\boxed{£}$	$\boxed{¤}$	$\boxed{¥}$	$\boxed{¦}$	$\boxed{§}$	¨	$\boxed{©}$	ª	$\boxed{«}$	$\boxed{¬}$		$\boxed{®}$	¯	
00B0	$\boxed{°}$	$\boxed{±}$	²	³	´	µ	$\boxed{¶}$	·	¸	¹	º	$\boxed{»}$	¼	½	¾	$\boxed{¿}$	XML disallows · initially
00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	
00D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	$\boxed{\times}$	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	
00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	
00F0	ð	ñ	ò	ó	ô	õ	ö	$\boxed{\div}$	ø	ù	ú	û	ü	ý	þ	ÿ	
0100	Latin Extended-A																
0180 0200	Latin Extended-B																

0250	IPA Extensions, Spacing Modifier Letters	
0300	Combining Diacritical Marks	XML disallows these initally
0370	Greek and Coptic	XML specifically disallows ;
0400	Cyrillic	
0500	Cyrillic Supplement, Armenian, Hebrew	
0600	Arabic	
0700	Syriac, Arabic Supplement, Thaana, NKo	
0800	Samaritan	
0900	Devanagari, Bengali	
0A00	Gurmukhi, Gujarati	
0B00	Oriya, Tamil	
0C00	Telugu, Kannada	
0D00	Malayalam, Sinhala	
0E00	Thai, Lao	
0F00	Tibetan	
1000	Myanmar, Georgian	
1100	Hangul Jamo	
1200 **1300**	Ethiopic	
1380	Ethiopic Supplement, Cherokee	
1400 **1600**	Unified Canadian Aboriginal Syllabics	
1680	Ogham, Runic	The Ogham block contains a script-specific space:
1700	Tagalog, Hanunoo, Buhid, Tagbanwa, Khmer	
1800	Mongolian, Unified Canadian Aboriginal Syllabics Extended	The Mongolian block contains a script-specific space: ᠎
1900	Limbu, Tai Le, New Tai Lue, Khmer Symbols	

1A00	Buginese, Tai Tham	
1B00	Balinese, Sundanese	
1C00	Lepcha, Ol Chiki, Vedic Extensions	
1D00	Phonetic Extensions, Phonetic Extensions Supplement	
1DC0	Combining Diacritical Marks Supplement	XML does **not** disallow these initially
1E00	Latin Extended Additional	
1F00	Greek Extended	
2000	**General Punctuation**	

2000																	
2010	‐	‑	‒	–	—	―	‖	‗	‘	’	‚	‛	“	”	„	‟	
2020	†	‡	•	‣	․	‥	…	‧									
2030	‰	‱	′	″	‴	‵	‶	‷	‸	‹	›	※	‼	‽	‾	‿	XML disallows ‿ initially
2040	⁀	⁁	⁂	⁃	⁄	⁅	⁆	⁇	⁈	⁉	⁊	⁋	⁌	⁍	⁎	⁏	XML disallows ⁀ initially
2050	⁐	⁑	⁒	⁓	⁔	⁕	⁖	⁗	⁘	⁙	⁚	⁛	⁜	⁝	⁞		
2060																	

2070	Superscripts and Subscripts	
20A0	Currency Symbols	
20D0	Combining Diacritical Marks for Symbols	XML does **not** disallow these initially
2100	Letterlike Symbols	
2150	Number Forms	
2190	Arrows	
2200	Mathematical Operators	
2300	Miscellaneous Technical	
2400	Control Pictures	
2440	Optical Character Recognition	
2460	Enclosed Alphanumerics	

2500	Box Drawing
2580	Block Elements
25A0	Geometric Shapes
2600	Miscellaneous Symbols
2700	Dingbats
2776	Dingbats (circled digits)
2794	Dingbats
27C0	Miscellaneous Mathematical Symbols-A
27F0	Supplemental Arrows-A
2800	Braille Patterns
2900	Supplemental Arrows-B
2980	Miscellaneous Mathematical Symbols-B
2A00	Supplemental Mathematical Operators
2B00	Miscellaneous Symbols and Arrows
2C00	Glagolitic, Latin Extended-C, Coptic
2D00	Georgian Supplement, Tifinagh, Ethiopic Extended, Cyrillic Extended-A
2E00	Supplemental Punctuation
2E80	CJK Radicals Supplement, Kangxi Radicals
2FF0	Ideographic Description
3000	**CJK Symbols and Punctuation**
3000	、 。 〃 〄 々 〆 〇 〈 〉 《 》 「 」 『 』
3010	【 】 〒 〓 〔 〕 〖 〗 〘 〙 〚 〛 〜 〝 〞 〟
3020	〠 〡 〢 〣 〤 〥 〦 〧 〨 〩 〪 〫 〬 〭 〮 〯
3030	〰 〱 〲 〳 〴 〵 〶 〷 〸 〹 〺 〻 〼 〽 〾 〿
3040	Hiragana, Katakana

3100	Bopomofo, Hangul Compatibility Jamo, Kanbun, Bopomofo Extended, CJK Strokes, Katakana Phonetic Extensions
3200	Enclosed CJK Letters and Months
3300	CJK Compatibility
3400 **4D00**	CJK Unified Ideographs Extension A
4DC0	Yijing Hexagram Symbols
4E00 **9F00**	CJK Unified Ideographs
A000 **A400**	Yi Syllables
A490	Yi Radicals, Lisu
A500 **A600**	Vai
A640	Cyrillic Extended-B, Bamum
A700	Modifier Tone Letters, Latin Extended-D
A800	Syloti Nagri, Common Indic Number Forms, Phags-pa, Saurashtra, Devanagari Extended
A900	Kayah Li, Rejang, Hangul Jamo Extended-A, Javanese
AA00	Cham, Myanmar Extended-A, Tai Viet
AB00	Meetei Mayek
AC00 **D700**	Hangul Syllables
D7B0	Hangul Jamo Extended-B
D800 **DB00**	High Surrogates
DB80	High Private Use Surrogates
DC00 **DF00**	Low Surrogates

E000 **F800**	Private Use Area	
F900 **FA00**	CJK Compatibility Ideographs	
FB00	Alphabetic Presentation Forms	
FB50 **FD00**	Arabic Presentation Forms-A	
FDD0	(non-characters)	
FDF0	Arabic Presentation Forms-A	This block contains Pattern_Syntax characters ﴾ and ﴿
FE00	Variation Selectors, Vertical Forms	
FE20	Combining Half Marks	XML does **not** disallow these initally
FE30	CJK Compatibility Forms	This block also contains Pattern_Syntax characters ﹅ and ﹆
FE50	Small Form Variants, Arabic Presentation Forms-B	
FF00	Halfwidth and Fullwidth Forms	
	Specials	
FFF0	�▯▯	

Beyond the BMP

The Supplementary Private Use Area extends from F0000 through 10FFFF; both [AltId] and [XML2008] disallow characters in that range.

In addition, [AltId] disallows, as non-characters, the last two code positions of each plane, i.e. every position of the form *P*FFFE or *P*FFFF, for any value of *P*.

Otherwise, no character outside the BMP is disallowed as an identifier character by either specification.