Stream:	Internet Engineering Task Force (IETF)	
RFC:	9839	
Category:	Standards Track	
Published:	August 2025	
ISSN:	2070-1721	
Authors:	T. Bray	P. Hoffman
	Textuality Services	*ICANN*

RFC 9839
Unicode Character Repertoire Subsets

Abstract

This document discusses subsets of the Unicode character repertoire for use in protocols and data formats and specifies three subsets recommended for use in IETF specifications.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9839.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 2
1.1. Notation 3
2. Characters and Code Points 3
2.1. Encoding Forms 4
2.2. Problematic Code Points 4
2.2.1. Surrogates 4
2.2.2. Control Codes 5
2.2.3. Noncharacters 5
3. Dealing with Problematic Code Points 5
4. Subsets 6
4.1. Unicode Scalars 6
4.2. XML Characters 7
4.3. Unicode Assignables 7
5. Using Subsets 8
6. IANA Considerations 8
7. Security Considerations 8
8. References 8
8.1. Normative References 8
8.2. Informative References 9
Acknowledgements 10
Authors' Addresses 10

1. Introduction

Protocols and data formats frequently contain or are made up of textual data. Such text is normally composed of Unicode [UNICODE] characters, to support use by speakers of many languages. Unicode characters are represented by numeric code points, and the "set of all Unicode code points" is generally not a good choice for use in text fields. Unicode recognizes different types of code points, not all of which are appropriate in protocols or even associated

with characters. Therefore, even if the desire is to support "all Unicode characters", a subset of the Unicode code point repertoire should be specified. Subsets such as those discussed in this document are appropriate choices when more-specific limitations do not apply.

In this document, "subset" means a subset of the Unicode character repertoire. This document specifies subsets that exclude some or all of the code points that are "problematic" as defined in Section 2.2. Authors should have a way to concisely and exactly reference a stable specification that identifies which subset a protocol or data format accepts.

This document discusses issues that apply in choosing subsets, names two subsets that have been popular in practice, and suggests one new subset. The intended use is to serve as a convenient target for cross-reference from other specifications whose authors wish to exclude problematic code points from the data format or protocol being specified.

Note that this document only provides guidance on avoiding the use of code points that cannot be used for interoperable interchange of Unicode textual data. Dealing with strings, particularly in the context of user interfaces, requires addressing language, text rendering direction, alternate representations of the same abstract character, and so on. These issues, among many others, led to efforts by the Unicode Consortium, efforts by the IETF such as [IDN] and [PRECIS], and internationalization efforts by W3C such as [W3C-CHAR]. The results of these efforts should be consulted by anyone engaging in such work.

1.1. Notation

In this document, the numeric values assigned to Unicode characters are provided in hexadecimal. This document uses Unicode's standard notation of "U+" followed by four or more hexadecimal digits. For example, "A", decimal 65, is expressed as U+0041, and "🖤" (Black Heart), decimal 128,420, is U+1F5A4.

Groups of numeric values described in Section 4 are given in ABNF [RFC5234]. In ABNF, hexadecimal values are preceded by "%x" rather than "U+".

All the numeric ranges in this document are inclusive.

The subsets are described in ABNF.

2. Characters and Code Points

Definition D9 in Section 3.4 of [UNICODE] defines "Unicode codespace" as "a range of integers from 0 to $10FFFF_{16}$". Definition D10 defines "code point" as "Any value in the Unicode codespace".

The Unicode Standard's definition of "Unicode character" is conceptual. However, each Unicode character is assigned a code point, used to represent the characters in computer memory and storage systems and to specify allowed subsets in specifications.

There are 1,114,112 ($17 * 2^{16}$) code points; as of Unicode 16.0 (2024), about 155,000 have been assigned to characters. Since unassigned code points regularly become assigned when new characters are added to Unicode, it is usually not a good practice to specify that unassigned code points should be avoided.

2.1. Encoding Forms

Unicode describes a variety of encoding forms that can be used to marshal code points into byte sequences. A survey of these is beyond the scope of this document. However, it is useful to note that "UTF-16" represents each code point with one or two 16-bit chunks, while "UTF-8" uses variable-length byte sequences [RFC3629].

The "IETF Policy on Character Sets and Languages", BCP 18 [RFC2277], says "Protocols **MUST** be able to use the UTF-8 charset", which becomes a mandate to use UTF-8 for any protocol or data format that specifies a single encoding form. UTF-8 is widely used for interoperable data formats such as JSON, YAML, CBOR, and XML.

2.2. Problematic Code Points

This section classifies all the code points that can never represent useful text and, in some cases, can lead to software misbehavior as "problematic". This is a low bar; the PRECIS [RFC8264] framework's "IdentifierClass" and "FreeformClass" exclude many more code points that can cause problems when displayed to humans, in some cases presenting security risks. Specifications of fields in protocols and data formats whose contents are designed for display to and interactions with humans would benefit from careful consideration of the issues described by PRECIS; its more-restrictive subsets might be better choices than those specified in this document.

Definition D10a in Section 3.4 of [UNICODE] defines seven code point types. Three types of code points are assigned to entities that are not actually characters or whose value as Unicode characters in text fields is questionable: "Surrogate", "Control", and "Noncharacter". In this document, "problematic" refers to code points whose type is "Surrogate" or "Noncharacter" and to "legacy controls" as defined in Section 2.2.2.2 below.

Definition D49 in [UNICODE] concerns the "private-use" type, and Section 3.5.10 states that they "are considered to be assigned characters". Section 23.5 further states that these characters' "use may be determined by private agreement among cooperating users". Because private-use code points may have uses based on private agreements, this document does not classify them as "problematic".

2.2.1. Surrogates

A total of 2,048 code points, in the range U+D800-U+DFFF, are divided into two blocks called "high surrogates" and "low surrogates"; collectively, the 2,048 code points are referred to as "surrogates". Section 23.6 of [UNICODE] specifies how surrogates may be used in Unicode texts encoded in UTF-16, where a high-surrogate/low-surrogate pair represents a code point greater than U+FFFF.

A surrogate that occurs in text encoded in any encoding form other than UTF-16 has no meaning. In particular, Section 3.9.3 of [UNICODE] forbids representing a surrogate in UTF-8.

2.2.2. Control Codes

Section 23.1 of [UNICODE] introduces the control codes for compatibility with legacy pre-Unicode standards. They comprise 65 code points in the ranges U+0000-U+001F ("C0 controls") and U+0080-U+009F ("C1 controls"), plus U+007F, "DEL".

2.2.2.1. Useful Controls

The C0 controls include newline (U+000A), carriage return (U+000D), and tab (U+0009); this document refers to these three characters as the "useful controls".

2.2.2.2. Legacy Controls

Aside from the useful controls, both the C0 and C1 control codes are mostly obsolete and generally lack interoperable semantics. This document uses the phrase "legacy controls" to describe control codes that are not useful controls.

Because the code points for C0 controls include the 32 smallest integers including zero, they are likely to occur in data as a result of programming errors.

2.2.3. Noncharacters

Certain code points are classified as "noncharacters", and [UNICODE] asserts repeatedly that they are not designed or used for open interchange.

Code points are organized into 17 "planes", each containing 2^{16} code points. The last two code points in each plane are noncharacters: U+FFFE, U+FFFF, U+1FFFE, U+1FFFF, U+2FFFE, U+2FFFF, and so on, up to U+10FFFE, U+10FFFF.

The code points in the range U+FDD0-U+FDEF are noncharacters.

3. Dealing with Problematic Code Points

"Maintaining Robust Protocols" [RFC9413] provides a thorough discussion of strategies for dealing with issues in input data.

Different types of problematic code points cause different issues. Noncharacters and legacy controls are unlikely to cause software failures, but they cannot usefully be displayed to humans, and they can be used in attacks based on attempting to display text that includes them.

The behavior of software that encounters surrogates is unpredictable and differs among programming-language implementations, even between different API calls in the same language.

Section 3.9 of [UNICODE] makes it clear that a UTF-8 byte sequence that would map to a surrogate is ill-formed. If a specification requires that input data be encoded with UTF-8, and if all input were well-formed, implementors would never have to concern themselves with surrogates.

Unfortunately, industry experience teaches that problematic code points, including surrogates, can and do occur in program input where the source of input data is not controlled by the implementor. In particular, the specification of JSON allows any code point to appear in object member names and string values [RFC8259].

For example, the following is a conforming JSON text:

```
{"example": "\u0000\u0089\uDEAD\uD9BF\uDFFF"}
```

The value of the "example" field contains the C0 control NUL, the C1 control "CHARACTER TABULATION WITH JUSTIFICATION", an unpaired surrogate, and the noncharacter U+7FFFF encoded per JSON rules as two escaped UTF-16 surrogate code points as described in Section 7 of [RFC8259]. It is unlikely to be useful as the value of a text field. That value cannot be serialized into well-formed UTF-8, but the behavior of libraries asked to parse the sample is unpredictable; some will silently parse this and generate an ill-formed UTF-8 string.

Two reasonable options for dealing with problematic input are either rejecting text containing problematic code points or replacing the problematic code points with placeholders.

Silently deleting an ill-formed part of a string is a known security risk. Responding to that risk, Section 3.2 of [UNICODE] recommends dealing with ill-formed byte sequences by signaling an error or replacing problematic code points, ideally with "" (U+FFFD, REPLACEMENT CHARACTER).

4. Subsets

This section describes three increasingly restrictive subsets that can be used in specifying acceptable content for text fields in protocols and data types. Specifications can refer to these subsets by the names "Unicode Scalars", "XML Characters", and "Unicode Assignables".

4.1. Unicode Scalars

Definition D76 in Section 3.9 of [UNICODE] defines the term "Unicode scalar value" as "Any Unicode code point except high-surrogate and low-surrogate code points".

The "Unicode Scalars" subset can be expressed as an ABNF production:

```
unicode-scalar =
   %x0-D7FF /    ; exclude surrogates
   %xE000-10FFFF
```

This subset is the default for Concise Binary Object Representation (CBOR) [RFC8949] and has the advantage of excluding surrogates. However, it includes legacy controls and noncharacters.

4.2. XML Characters

The XML 1.0 Specification (Fifth Edition) [XML], in its grammar production labeled "Char", specifies a subset of Unicode code points that excludes surrogates, legacy C0 controls, and the noncharacters U+FFFE and U+FFFF.

The "XML Characters" subset can be expressed as an ABNF production:

```
xml-character =
   %x9 / %xA / %xD /    ; useful controls
   %x20-D7FF /          ; exclude surrogates
   %xE000-FFFD /        ; exclude FFFE and FFFF nonchars
   %x10000-10FFFF
```

While this subset does not exclude all the problematic code points, the C1 controls are less likely than the C0 controls to appear erroneously in data and have not been observed to be a frequent source of problems. Also, the noncharacters greater in value than U+FFFF are rarely encountered.

4.3. Unicode Assignables

This document defines the "Unicode Assignables" subset as all the Unicode code points that are not problematic. This, a proper subset of each of the others, comprises all code points that are currently assigned, excluding legacy control codes, or that might be assigned in the future.

Unicode Assignables can be expressed as an ABNF production:

```
unicode-assignable =
   %x9 / %xA / %xD /                ; useful controls
   %x20-7E /                        ; exclude C1 controls and DEL
   %xA0-D7FF /                      ; exclude surrogates
   %xE000-FDCF /                    ; exclude FDD0 nonchars
   %xFDF0-FFFD /                    ; exclude FFFE and FFFF nonchars
   %x10000-1FFFD / %x20000-2FFFD / ; (repeat per plane)
   %x30000-3FFFD / %x40000-4FFFD /
   %x50000-5FFFD / %x60000-6FFFD /
   %x70000-7FFFD / %x80000-8FFFD /
   %x90000-9FFFD / %xA0000-AFFFD /
   %xB0000-BFFFD / %xC0000-CFFFD /
   %xD0000-DFFFD / %xE0000-EFFFD /
   %xF0000-FFFFD / %x100000-10FFFD
```

5. Using Subsets

Many IETF specifications rely on well-known data formats such as JSON, Internet JSON (I-JSON), CBOR, YAML, and XML. These formats specify default subsets. For example, JSON allows object member names and string values to include any Unicode code point, including all the problematic types.

A protocol based on JSON can be made more robust and implementor-friendly by restricting the contents of object member names and string values to one of the subsets described in Section 4. Equivalent restrictions are possible for other packaging formats such as I-JSON, XML, YAML, and CBOR.

Note that escaping techniques such as those in the JSON example in Section 3 cannot be used to circumvent this sort of restriction, which applies to data content, not textual representation in packaging formats. If a specification restricted a JSON field value to the Unicode Assignables, the example would remain a conforming JSON text but the data it represents would not constitute Unicode Assignable code points.

6. IANA Considerations

This document has no IANA actions.

7. Security Considerations

Section 3 of this document discusses security issues.

Unicode Security Considerations [TR36] is a wide-ranging survey of the issues implementors should consider while writing software to process Unicode text. Unicode Source Code Handling [TR55] discusses use of Unicode in programming languages, with a focus on security issues. Many of the attacks they discuss are aimed at deceiving human readers, but vulnerabilities involving issues such as surrogates and noncharacters are also covered and, in fact, can contribute to human-deceiving exploits.

The security considerations in Section 12 of [RFC8264] generally apply to this document as well.

Note that the Unicode-character subsets specified in this document are increasingly restrictive, omitting more and more problematic code points, and thus should be less and less susceptible to many of these exploits. The subset in Section 4.3, "Unicode Assignables", excludes all of these code points.

8. References

8.1. Normative References

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[TR36] Davis, M., Ed. and M. Suignard, Ed., "Unicode Security Considerations", <https://www.unicode.org/reports/tr36/>.

[TR55] Leroy, R., Ed. and M. Davis, Ed., "Unicode Source Code Handling", <https://www.unicode.org/reports/tr55/>.

[UNICODE] The Unicode Consortium, "The Unicode Standard", <http://www.unicode.org/versions/latest/>. Note that this reference is to the latest version of Unicode, rather than to a specific release. It is not expected that future changes in the Unicode Standard will affect the referenced definitions.

8.2. Informative References

[IDN] "Internationalized Domain Name Working Group", <https://datatracker.ietf.org/group/idn/>.

[PRECIS] "PRECIS Working Group", <https://datatracker.ietf.org/group/precis/>.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <https://www.rfc-editor.org/info/rfc2277>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9413] Thomson, M. and D. Schinazi, "Maintaining Robust Protocols", RFC 9413, DOI 10.17487/RFC9413, June 2023, <https://www.rfc-editor.org/info/rfc9413>.

[W3C-CHAR] W3C, "Character encodings: Essential concepts", <https://www.w3.org/International/articles/definitions-characters/>.

[XML] Bray, T., Ed., Paoli, J., Ed., McQueen, C.M., Ed., Maler, E., Ed., and F. Yergeau, Ed., "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation, 26 November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126/>.

Acknowledgements

Thanks are due to Guillaume Fortin-Debigaré, who filed an errata report against RFC 8259, "The JavaScript Object Notation (JSON) Data Interchange Format", noting frequent references to "Unicode characters", when in fact the RFC formally specifies the use of Unicode code points.

Thanks also to Asmus Freytag for careful review and many constructive suggestions aimed at making the language more consistent with the structure of the Unicode Standard.

Thanks also to James Manger for the correctness of the ABNF and JSON samples.

Thanks also to Addison Phillips and the W3C Internationalization Working Group for helpful suggestions on language and references.

Thoughtful comments during the many draft versions of this document, which helped tighten up wording and make difficult points clearer, were contributed by Harald Alvestrand, Martin J. Dürst, Donald E. Eastlake, John Klensin, Barry Leiba, Glyn Normington, Peter Saint-Andre, and Rob Sayre.

Authors' Addresses

Tim Bray
Textuality Services
Email: tbray@textuality.com

Paul Hoffman
ICANN
Email: paul.hoffman@icann.org