Abstract

This memo defines the application/www-form-urlencoded format, a compact data format that encodes ordered data sets of name-value pairs of character data. The format is similar to the format application/x-www-form-urlencoded first defined in RFC 1866, but addresses some of that format’s shortcomings.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 29, 2011.

Copyright Notice

Copyright (c) 2010 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 (http://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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
# Table of Contents

1. Introduction ................................................. 3  
2. Terminology and Conformance ................................. 3  
3. Format syntax .............................................. 4  
4. Format semantics ........................................... 4  
5. Examples ................................................... 5  
6. Security considerations ..................................... 7  
7. IANA Considerations ......................................... 7  
8. Media type registration ..................................... 8  
9. References .................................................. 8  
  9.1. Normative References ................................... 8  
  9.2. Informative References .................................. 9  

**Appendix A.** Acknowledgements ................................. 9
1. Introduction

RFC 1866 [RFC1866] introduced the application/x-www-form-urlencoded media type to facilitate the encoding and transmission of form data sets. Formats based on RFC 1866 continued to use this media type as default encoding format, and other protocols adopted the type for similar purposes. The format defined in this document addresses some of the RFC 1866 format’s shortcomings.

The application/www-form-urlencoded format defined in this document encodes ordered data sets of pairs consisting of a name and a (possibly undefined) value as a string, with pairs separated by semicolons and names and values separated by the equals sign. Special characters are escaped using the percent-encoding scheme also used for resource identifiers. Issues of internationalization are addressed through the use of the UTF-8 character encoding scheme.

For compatibility with the RFC 1866 format the ampersand character is tolerated as alternative separator character, and the plus sign may be used to represent space characters. The new format accepts any string as valid representation of a data set, except for character encoding errors, in keeping with typical implementations of the RFC 1866 format.

2. Terminology and Conformance

A character string is a sequence of Unicode scalar values. An octet string is a sequence of octets.

A character string conforms to this specification if and only if encoding it using the UTF-8 character encoding yields an octet string that conforms to this specification.

A octet string conforms to this specification if and only if it is, after replacing all sequences that match pct-encoded [RFC3986] by the corresponding octets, a valid UTF-8 sequence.

A software module that encodes data sets into character strings conforms to this specification if and only if it does so as defined in section 3.

A software module that decodes character or octet strings into data sets conforms to this specification if and only if it does so as defined in section 3.
3. Format syntax

The syntax of the application/www-form-urlencoded format is defined by the following ABNF [RFC5234] grammar. The grammar is ambiguous: the empty string matches both 'empty-set' and 'pairs' and percent-encoded sequences match 'escape' and 'percent' followed by other characters. A match for 'escape' takes precedence over a match involving 'percent'. The choice between interpreting the empty string as an empty data set or a pair consisting of the empty string as name and an undefined value is made by individual applications.

```
data-set  = empty-set / pairs
pairs     = pair *(seperator pair)
pair      = name [ "=" value ]
name      = *(namechar / escape / percent / plus)
value     = *(valuechar / escape / percent / plus)
nametchar = <any octet except ";", ";", "+", ";", ";">  
valuechar = <any octet except ";", ";", "+", ";">  
escape    = "\%" 2hexdig  
separator = ";" / ";"  
percent   = ";"  
plus      = "+"  
empty-set = ""
```

A character string is decoded by encoding it using the UTF-8 character encoding and then decoding the resulting octet string. An octet string is decoded by replacing any instance of 'escape' by the corresponding octet, replacing any instance of 'plus' by the U+0020 SPACE character, and then decoding the resulting 'name' and 'value' instances using the UTF-8 character encoding. If that results in an error, the data set is malformed and represents nothing.

A data set is encoded by encoding the names and values using the UTF-8 character encoding, replacing any octet not matching 'namechar' in the names and replacing any octet not matching 'valuechar' in the values by their percent-encoded equivalent and concatenating them using "=" and ";" as separators. The ampersand can be used as alternative separator, but doing so is discouraged. Similarly, ";" only has to be escaped when it is followed by two hex digits, but keeping it unescaped is discouraged. Spaces may additionally be replaced by the plus sign. Implementations are free to percent-encode additional octets.

4. Format semantics

This specification defines only the mapping between data sets and their encoded form. It is up to individual applications using this format to define, for instance, whether the ordering of pairs is...
significant or how multiple pairs with the same name are handled.

5. Examples

This section provides a number of examples that illustrate encoding and decoding of data sets as defined in this specification. At the beginning of each example is the data set under consideration; it is followed by equivalent encoded data sets (==) and different ones (!!). The notation <U+XXXX> is used to refer to Unicode scalar values. The equivalence rules here are only those that all implementations must recognize, individual applications may define additional rules.

There are multiple ways to represent space characters, they can occur literally, as a plus sign, or as percent-encoded sequences. All white space is considered significant and retained unmodified.

```
[[' a ', ' 1 '])
== ' a = 1 '
== '+a+=+1+'
== '%20a%20=%201%20'
!! 'a=1'
```

Characters typically used to represent the end of a line are not considered special, and no normalization of such characters is performed.

```
[['text', 'x<U+000A>y')]
== 'text=x<U+000A>y'
== 'text=x%0Ay'
!! 'text=x%0D%0Ay'
!! 'text=x%0Dy'
```

Similarly, characters outside the repertoire of US-ASCII are not handled in any special manner:

```
[['constellation', 'Bo<U+00F6>tes')]
== 'constellation=Bo<U+00F6>tes'
== 'constellation=Bo%C3%B6tes'
!! 'constellation=Boo<U+0308>tes'
```

The character U+0000 can occur in data sets and encoders and decoders have to be prepared to handle them unless applications that employ them guarantee otherwise. It is incorrect so truncate the data set at the first occurrence of such a character.
The following example illustrates handling of percent-encoding. While it is discouraged to have percent signs in encoded data sets that are not followed by two hex digits, decoders have to be prepared to handle them.

[['Cipher', {'cipher': 'c=(m^e)\%n'}]]
== 'Cipher=c%3D(m%5Ee)%25n'
== 'Cipher=c=(m%5Ee)%25n'
== '%43%69%70%68%65%72=%63%3D%28%6D%5E%65%29%25%6e'
!! 'Cipher%3Dc%3D(m%5Ee)'
!! 'Cipher=c'

The following six examples illustrate handling of empty name fields, empty value fields, and undefined value fields. The empty string is ambiguous as noted earlier in this document.

[['', undefined], ('', undefined)] == ';
[['', undefined], ('', '')]        == ';='
[['', ''), ('', undefined)]        == '=;'
[['', ''), ('', '')]               == '=;='
[['', undefined]]                  == ''
[[]]                                 == ''
[['', ',']]                        == '=

The separator characters ";" and ";" can both be used in encoded data sets; they always separate pairs if not escaped, even if both of them occur in a single string.

[['a&b', '1'], ('c', '2;3'), ('e', '4')]  
== 'a%26b=1;c=2%3B3;e=4'
== 'a%26b=1&c=2%3B3&e=4'
== 'a%26b=1;c=2%3B3&e=4'
== 'a%26b=1&c=2%3B3;e=4'
!! 'a&b=1;c=2%3B3;e=4'
!! 'a%26b=1&c=2;3&e=4'

Undefined values allow to represent certain information in a more compact form. A filter that selects columns in a product listing for instance could be encoded as follows:

[['image', undefined], ('title', undefined), ('price', undefined)]
The following examples do not conform to this specification due to character encoding errors and consequently represent nothing.

* 'Lookup=%ED%AD%80%ED%B1%BF'
* 'Lookup=%FE%83%9E%AB%9B%BB%AF'
* 'Lookup=%C0%80'
* 'Lookup=%C3'
* 'Lookup=Bo%F6tes'

6. Security considerations

None not already inherent to the processing of the UTF-8 character encoding [RFC3629] and the handling of percent-encoded sequences [RFC3986]. Depending on how the format defined in this document is being used, the security considerations of the aforementioned RFCs, [RFC3987], and [RFC3875] might inform security decisions.

7. IANA Considerations

This memo registers application/www-form-urlencoded as per [RFC4288].
8. Media type registration

Type name: application
Subtype name: www-form-urlencoded
Required parameters: none
Optional parameters: none

Note: The media type does not have a 'charset' parameter, it is incorrect to specify one and to associate any significance to it if specified. The character encoding is always UTF-8. The Unicode encoding form signature is not supported; a leading U+FEFF character will be considered part of a <name>.

Encoding considerations: 8bit

Interoperability considerations:
  None, except as noted in other sections of this document.

Published specification: RFC XXXX
Applications that use this media type:
  Systems that interchange data sets of name-value pairs.

Additional information:

  Magic number(s):          n/a
  File extension(s):        n/a
  Macintosh file type code(s): TEXT
  Fragment identifiers:     n/a

Person & email address to contact for further information:
  See Author’s Address section.

Intended usage: COMMON
Restrictions on usage: n/a
Author: See Author’s Address section.
Change controller: The IESG.

9. References

9.1. Normative References


9.2. Informative References


Appendix A. Acknowledgements

Mark Nottingham pointed out a serious omission in the first draft of this document.

Author’s Address

Bjoern Hoehrmann
Mittelstrasse 50
39114 Magdeburg
Germany

EMail: mailto:bjoern@hoehrmann.de
URI: http://bjoern.hoehrmann.de

Note: Please write "Bjoern Hoehrmann" with o-umlaut (U+00F6) wherever possible, e.g., as "Bjöhn Hörmann" in HTML and XML.