Internationalized Resource Identifiers (iri)
Internet-Draft
Obsoletes: RFC 3987
(if approved)
Intended status: Standards Track
Expires: August 2, 2010

M. Duerst
Aoyama Gakuin University
M. Suignard
Unicode Consortium
L. Masinter
Adobe
January 29, 2010

Internationalized Resource Identifiers (IRIs) draft-ietf-iri-3987bis-00

Abstract

This document defines the Internationalized Resource Identifier (IRI) protocol element, as an extension of the Uniform Resource Identifier (URI). An IRI is a sequence of characters from the Universal Character Set (Unicode/ISO 10646). Grammar and processing rules are given for IRIs and related syntactic forms.

In addition, this document provides named additional rule sets for processing otherwise invalid IRIs, in a way that supports other specifications that wish to mandate common behavior for 'error' handling. In particular, rules used in some XML languages (LEIRI) and web applications are given.

Defining IRI as new protocol element (rather than updating or extending the definition of URI) allows independent orderly transitions: other protocols and languages that use URIs must explicitly choose to allow IRIs.

Guidelines are provided for the use and deployment of IRIs and related protocol elements when revising protocols, formats, and software components that currently deal only with URIs.

[RFC Editor: Please remove this paragraph before publication.] This document is intended to update RFC 3987 and move towards IETF Draft Standard. This version is essentially identical to draft-duerst-iri-bis-07.txt, and is submitted as an initial draft to start WG discussions. For discussion and comments on this draft, please join the IETF IRI WG by subscribing to the mailing list public-iri@w3.org.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 2, 2010.

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 BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 5
1.1. Overview and Motivation 5
1.2. Applicability 6
1.3. Definitions 6
1.4. Notation 9
2. IRI Syntax 9
2.1. Summary of IRI Syntax 10
2.2. ABNF for IRI References and IRIs 10
3. Processing IRIs and related protocol elements 13
3.1. Converting to UCS 14
3.2. Parse the IRI into IRI components 14
3.3. General percent-encoding of IRI components 15
3.4. Mapping ireg-name 15
3.5. Mapping query components 17
3.6. Mapping IRIs to URIs 17
3.7. Converting URIs to IRIs 17
3.7.1. Examples 19
4. Bidirectional IRIs for Right-to-Left Languages 20
4.1. Logical Storage and Visual Presentation 21
4.2. Bidi IRI Structure 22
4.3. Input of Bidi IRIs 23
4.4. Examples 23
5. Normalization and Comparison 25
5.1. Equivalence 25
5.2. Preparation for Comparison 26
5.3. Comparison Ladder 27
5.3.1. Simple String Comparison 27
5.3.2. Syntax-Based Normalization 28
5.3.3. Scheme-Based Normalization 31
5.3.4. Protocol-Based Normalization 32
6. Use of IRIs 33
6.1. Limitations on UCS Characters Allowed in IRIs 33
6.2. Software Interfaces and Protocols 33
6.3. Format of URIs and IRIs in Documents and Protocols 33
6.4. Use of UTF-8 for Encoding Original Characters 34
6.5. Relative IRI References 36
7. Liberal handling of otherwise invalid IRIs 36
7.1. LEIRI processing 36
7.2. Web Address processing 36
7.3. Characters not allowed in IRIs 38
8. URI/IRI Processing Guidelines (Informative) 40
8.1. URI/IRI Software Interfaces 40
8.2. URI/IRI Entry 41
8.3. URI/IRI Transfer between Applications 42
8.4. URI/IRI Generation 42
8.5. URI/IRI Selection 43

8.6. Display of URIs/IRIs 43
8.7. Interpretation of URIs and IRIs 44
8.8. Upgrading Strategy 44
9. IANA Considerations 45
10. Security Considerations 46
11. Acknowledgements 47
12. Open Issues 48
13. Change Log 50
13.1. Changes from draft-duerst-iri-bis-07 to draft-ietf-iri-3987bis-00 50
13.2. Changes from -06 to -07 of draft-duerst-iri-bis 50
13.2.1. OLD WAY 50
13.2.2. NEW WAY 51
13.3. Changes from -05 to -06 of draft-duerst-iri-bis 51
13.4. Changes from -04 to -05 of draft-duerst-iri-bis 51
13.5. Changes from -03 to -04 of draft-duerst-iri-bis 51
13.6. Changes from -02 to -03 of draft-duerst-iri-bis 52
13.7. Changes from -01 to -02 of draft-duerst-iri-bis 52
13.8. Changes from -00 to -01 of draft-duerst-iri-bis 52
13.9. Changes from RFC 3987 to -00 of draft-duerst-iri-bis . . . 52
14. References 52
14.1. Normative References 52
14.2. Informative References 53
Appendix A. Design Alternatives 56
A.1. New Scheme(s) 56
A.2. Character Encodings Other Than UTF-8 56
A.3. New Encoding Convention 56
A.4. Indicating Character Encodings in the URI/IRI 57
Authors' Addresses 57

1. Introduction

1.1. Overview and Motivation

A Uniform Resource Identifier (URI) is defined in [RFC3986] as a sequence of characters chosen from a limited subset of the repertoire of US-ASCII [ASCII] characters.

The characters in URIs are frequently used for representing words of natural languages. This usage has many advantages: Such URIs are easier to memorize, easier to interpret, easier to transcribe, easier to create, and easier to guess. For most languages other than English, however, the natural script uses characters other than A - Z. For many people, handling Latin characters is as difficult as handling the characters of other scripts is for those who use only the Latin alphabet. Many languages with non-Latin scripts are transcribed with Latin letters. These transcriptions are now often used in URIs, but they introduce additional difficulties.

The infrastructure for the appropriate handling of characters from additional scripts is now widely deployed in operating system and application software. Software that can handle a wide variety of scripts and languages at the same time is increasingly common. Also, an increasing number of protocols and formats can carry a wide range of characters.

URIs are used both as a protocol element (for transmission and processing by software) and also a presentation element (for display and handling by people who read, interpret, coin, or guess them). The transition between these roles is more difficult and complex when dealing with the larger set of characters than allowed for URIs in [RFC3986].

This document defines the protocol element called Internationalized Resource Identifier (IRI), which allow applications of URIs to be extended to use resource identifiers that have a much wider repertoire of characters. It also provides corresponding "internationalized" versions of other constructs from [RFC3986], such as URI references. The syntax of IRIs is defined in Section 2.

Using characters outside of A - Z in IRIs adds a number of difficulties. Section 4 discusses the special case of bidirectional IRIs using characters from scripts written right-to-left. Section 5 discusses various forms of equivalence between IRIs. Section 6 discusses the use of IRIs in different situations. Section 8 gives additional informative guidelines. Section 10 discusses IRI-specific security considerations.

1.2. Applicability

IRIs are designed to allow protocols and software that deal with URIs to be updated to handle IRIs. A "URI scheme" (as defined by [RFC3986] and registered through the IANA process defined in [RFC4395] also serves as an "IRI scheme". Processing of IRIs is accomplished by extending the URI syntax while retaining (and not expanding) the set of "reserved" characters, such that the syntax for any URI scheme may be uniformly extended to allow non-ASCII characters. In addition, following parsing of an IRI, it is possible to construct a corresponding URI by first encoding characters outside of the allowed URI range and then reassembling the components.

Practical use of IRIs forms in place of URIs forms depends on the following conditions being met:

a. A protocol or format element MUST be explicitly designated to be able to carry IRIs. The intent is to avoid introducing IRIs into contexts that are not defined to accept them. For example, XML schema [XMLSchema] has an explicit type "anyURI" that includes IRIs and IRI references. Therefore, IRIs and IRI references can be in attributes and elements of type "anyURI". On the other hand, in the [RFC2616] definition of HTTP/1.1, the Request URI is defined as a URI, which means that direct use of IRIs is not allowed in HTTP requests.

b. The protocol or format carrying the IRIs MUST have a mechanism to represent the wide range of characters used in IRIs, either natively or by some protocol- or format-specific escaping mechanism (for example, numeric character references in [XML1]).

c. The URI scheme definition, if it explicitly allows a percent sign ("%") in any syntactic component, SHOULD define the interpretation of sequences of percent-encoded octets (using "%XX" hex octets) as octet from sequences of UTF-8 encoded strings; this is recommended in the guidelines for registering new schemes, [RFC4395]. For example, this is the practice for IMAP URLs [RFC2192], POP URLs [RFC2384] and the URN syntax [RFC2141]). Note that use of percent-encoding may also be restricted in some situations, for example, URI schemes that disallow percent-encoding might still be used with a fragment identifier which is percent-encoded (e.g., [XPointer]). See Section 6.4 for further discussion.

1.3. Definitions

The following definitions are used in this document; they follow the terms in [RFC2130], [RFC2277], and [ISO10646].

character: A member of a set of elements used for the organization, control, or representation of data. For example, "LATIN CAPITAL LETTER A" names a character.

octet: An ordered sequence of eight bits considered as a unit.

character repertoire: A set of characters (set in the mathematical sense).

sequence of characters: A sequence of characters (one after another).

sequence of octets: A sequence of octets (one after another).

character encoding: A method of representing a sequence of characters as a sequence of octets (maybe with variants). Also, a method of (unambiguously) converting a sequence of octets into a sequence of characters.

charset: The name of a parameter or attribute used to identify a character encoding.

UCS: Universal Character Set. The coded character set defined by ISO/IEC 10646 [ISO10646] and the Unicode Standard [UNIV4].

IRI reference: Denotes the common usage of an Internationalized Resource Identifier. An IRI reference may be absolute or relative. However, the "IRI" that results from such a reference only includes absolute IRIs; any relative IRI references are resolved to their absolute form. Note that in [RFC2396] URIs did not include fragment identifiers, but in [RFC3986] fragment identifiers are part of URIs.

URL: The term "URL" was originally used [RFC1738] for roughly what is now called a "URI". Books, software and documentation often refers to URIs and IRIs using the "URL" term. Some usages restrict "URL" to those URIs which are not URNs. Because of the ambiguity of the term using the term "URL" is NOT RECOMMENDED in formal documents.

LEIRI (Legacy Extended IRI) processing: This term was used in various XML specifications to refer to strings that, although not valid IRIs, were acceptable input to the processing rules in Section 7.1.

(Web Address, Hypertext Reference, HREF): These terms have been added in this document for convenience, to allow other specifications to refer to those strings that, although not valid IRIs, are acceptable input to the processing rules in Section 7.2. This usage corresponds to the parsing rules of some popular web browsing applications. ISSUE: Need to find a good name/ abbreviation for these.

running text: Human text (paragraphs, sentences, phrases) with syntax according to orthographic conventions of a natural language, as opposed to syntax defined for ease of processing by machines (e.g., markup, programming languages).

protocol element: Any portion of a message that affects processing of that message by the protocol in question.

presentation element: A presentation form corresponding to a protocol element; for example, using a wider range of characters.

create (a URI or IRI): With respect to URIs and IRIs, the term is used for the initial creation. This may be the initial creation of a resource with a certain identifier, or the initial exposition of a resource under a particular identifier.

generate (a URI or IRI): With respect to URIs and IRIs, the term is used when the identifier is generated by derivation from other information.

parsed URI component: When a URI processor parses a URI (following the generic syntax or a scheme-specific syntax, the result is a set of parsed URI components, each of which has a type (corresponding to the syntactic definition) and a sequence of URI characters.

parsed IRI component: When an IRI processor parses an IRI directly, following the general syntax or a scheme-specific syntax, the result is a set of parsed IRI components, each of which has a type (corresponding to the syntactice definition) and a sequence of IRI characters. (This definition is analogous to "parsed URI component".)

IRI scheme: A URI scheme may also be known as an "IRI scheme" if the scheme's syntax has been extended to allow non-US-ASCII characters according to the rules in this document.

1.4. Notation

RFCs and Internet Drafts currently do not allow any characters outside the US-ASCII repertoire. Therefore, this document uses various special notations to denote such characters in examples.

In text, characters outside US-ASCII are sometimes referenced by using a prefix of 'U+', followed by four to six hexadecimal digits.

To represent characters outside US-ASCII in examples, this document uses two notations: 'XML Notation' and 'Bidi Notation'.

XML Notation uses a leading '&#x', a trailing ';', and the hexadecimal number of the character in the UCS in between. For example, я stands for CYRILLIC CAPITAL LETTER YA. In this notation, an actual '&' is denoted by '&'.

Bidi Notation is used for bidirectional examples: Lower case letters stand for Latin letters or other letters that are written left to right, whereas upper case letters represent Arabic or Hebrew letters that are written right to left.

To denote actual octets in examples (as opposed to percent-encoded octets), the two hex digits denoting the octet are enclosed in "<" and ">". For example, the octet often denoted as 0xc9 is denoted here as <c9>.

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119].

2. IRI Syntax

This section defines the syntax of Internationalized Resource Identifiers (IRIs).

As with URIs, an IRI is defined as a sequence of characters, not as a sequence of octets. This definition accommodates the fact that IRIs may be written on paper or read over the radio as well as stored or transmitted digitally. The same IRI might be represented as different sequences of octets in different protocols or documents if these protocols or documents use different character encodings (and/or transfer encodings). Using the same character encoding as the containing protocol or document ensures that the characters in the IRI can be handled (e.g., searched, converted, displayed) in the same way as the rest of the protocol or document.

2.1. Summary of IRI Syntax

IRIs are defined by extending the URI syntax in [RFC3986], but extending the class of unreserved characters by adding the characters of the UCS (Universal Character Set, [ISO10646]) beyond U+007F, subject to the limitations given in the syntax rules below and in Section 6.1.

The syntax and use of components and reserved characters is the same as that in [RFC3986]. Each "URI scheme" thus also functions as an "IRI scheme", in that scheme-specific parsing rules for URIs of a scheme are be extended to allow parsing of IRIs using the same parsing rules.

All the operations defined in [RFC3986], such as the resolution of relative references, can be applied to IRIs by IRI-processing software in exactly the same way as they are for URIs by URI-processing software.

Characters outside the US-ASCII repertoire MUST NOT be reserved and therefore MUST NOT be used for syntactical purposes, such as to delimit components in newly defined schemes. For example, U+00A2, CENT SIGN, is not allowed as a delimiter in IRIs, because it is in the 'iunreserved' category. This is similar to the fact that it is not possible to use '-' as a delimiter in URIs, because it is in the 'unreserved' category.

2.2. ABNF for IRI References and IRIs

An ABNF definition for IRI references (which are the most general concept and the start of the grammar) and IRIs is given here. The syntax of this ABNF is described in [STD68]. Character numbers are taken from the UCS, without implying any actual binary encoding. Terminals in the ABNF are characters, not octets.

The following grammar closely follows the URI grammar in [RFC3986], except that the range of unreserved characters is expanded to include UCS characters, with the restriction that private UCS characters can occur only in query parts. The grammar is split into two parts: Rules that differ from [RFC3986] because of the above-mentioned expansion, and rules that are the same as those in [RFC3986]. For rules that are different than those in [RFC3986], the names of the non-terminals have been changed as follows. If the non-terminal contains 'URI', this has been changed to 'IRI'. Otherwise, an 'i' has been prefixed.

The following rules are different from those in [RFC3986]:

```
   IRI            = scheme ":" ihier-part [ "?" iquery ]
                    [ "#" ifragment ]

   ihier-part     = "//" iauthority ipath-abempty
                  / ipath-absolute
                  / ipath-rootless
                  / ipath-empty

   IRI-reference  = IRI / irelative-ref

   absolute-IRI   = scheme ":" ihier-part [ "?" iquery ]

   irelative-ref  = irelative-part [ "?" iquery ] [ "#" ifragment ]

   irelative-part = "//" iauthority ipath-abempty
                  / ipath-absolute
                  / ipath-noscheme
                  / ipath-empty

   iauthority     = [ iuserinfo "@" ] ihost [ ":" port ]
   iuserinfo      = *( iunreserved / pct-form / sub-delims / ":" )
   ihost          = IP-literal / IPv4address / ireg-name

   pct-form       = pct-encoded

   ireg-name      = *( iunreserved / sub-delims )

   ipath          = ipath-abempty   ; begins with "/" or is empty
                  / ipath-absolute  ; begins with "/" but not "//"
                  / ipath-noscheme  ; begins with a non-colon segment
                  / ipath-rootless  ; begins with a segment
                  / ipath-empty     ; zero characters

   ipath-abempty  = *( path-sep isegment )
   ipath-absolute = path-sep [ isegment-nz *( path-sep isegment ) ]
   ipath-noscheme = isegment-nz-nc *( path-sep isegment )
   ipath-rootless = isegment-nz *( path-sep isegment )
   ipath-empty    = 0<ipchar>
   path-sep       = "/"

   isegment       = *ipchar
   isegment-nz    = 1*ipchar
   isegment-nz-nc = 1*( iunreserved / pct-form / sub-delims
                        / "@" )
                  ; non-zero-length segment without any colon ":"

   ipchar         = iunreserved / pct-form / sub-delims / ":"
                  / "@"
```

```
   iquery         = *( ipchar / iprivate / "/" / "?" )

   ifragment      = *( ipchar / "/" / "?" / "#" )

   iunreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~" / ucschar

   ucschar        = %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF
                  / %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD
                  / %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD
                  / %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD
                  / %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD
                  / %xD0000-DFFFD / %xE1000-EFFFD

   iprivate       = %xE000-F8FF / %xE0000-E0FFF / %xF0000-FFFFD
                  / %x100000-10FFFD
```

Some productions are ambiguous. The "first-match-wins" (a.k.a. "greedy") algorithm applies. For details, see [RFC3986].

The following rules are the same as those in [RFC3986]:

```
scheme         = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )

port           = *DIGIT

IP-literal     = "[" ( IPv6address / IPvFuture  ) "]"

IPvFuture      = "v" 1*HEXDIG "." 1*( unreserved / sub-delims / ":" )

IPv6address    =                            6( h16 ":" ) ls32
               /                       "::" 5( h16 ":" ) ls32
               / [               h16 ] "::" 4( h16 ":" ) ls32
               / [ *1( h16 ":" ) h16 ] "::" 3( h16 ":" ) ls32
               / [ *2( h16 ":" ) h16 ] "::" 2( h16 ":" ) ls32
               / [ *3( h16 ":" ) h16 ] "::"    h16 ":"   ls32
               / [ *4( h16 ":" ) h16 ] "::"              ls32
               / [ *5( h16 ":" ) h16 ] "::"              h16
               / [ *6( h16 ":" ) h16 ] "::"

h16            = 1*4HEXDIG
ls32           = ( h16 ":" h16 ) / IPv4address

IPv4address    = dec-octet "." dec-octet "." dec-octet "." dec-octet

dec-octet      = DIGIT                 ; 0-9
               / %x31-39 DIGIT         ; 10-99
               / "1" 2DIGIT            ; 100-199
               / "2" %x30-34 DIGIT     ; 200-249
               / "25" %x30-35          ; 250-255

pct-encoded    = "%" HEXDIG HEXDIG

unreserved     = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved       = gen-delims / sub-delims
gen-delims     = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims     = "!" / "$" / "&" / "'" / "(" / ")"
               / "*" / "+" / "," / ";" / "="
```

This syntax does not support IPv6 scoped addressing zone identifiers.

3. Processing IRIs and related protocol elements

IRIs are meant to replace URIs in identifying resources within new versions of protocols, formats, and software components that use a UCS-based character repertoire. Protocols and components may use and process IRIs directly. However, there are still numerous systems and

protocols which only accept URIs or components of parsed URIs; that is, they only accept sequences of characters within the subset of US-ASCII characters allowed in URIs.

This section defines specific processing steps for IRI consumers which establish the relationship between the string given and the interpreted derivatives. These processing steps apply to both IRIs and IRI references (i.e., absolute or relative forms); for IRIs, some steps are scheme specific.

3.1. Converting to UCS

Input that is already in a Unicode form (i.e., a sequence of Unicode characters or an octet-stream representing a Unicode-based character encoding such as UTF-8 or UTF-16) should be left as is and not normalized (see (see Section 5.3.2.2).

If the IRI or IRI reference is an octet stream in some known non-Unicode character encoding, convert the IRI to a sequence of characters from the UCS; this sequence SHOULD also be normalized according to Unicode Normalization Form C (NFC, [UTR15]). In this case, retain the original character encoding as the "document character encoding". (DESIGN QUESTION: NOT WHAT MOST IMPLEMENTATIONS DO, CHANGE?)

In other cases (written on paper, read aloud, or otherwise represented independent of any character encoding) represent the IRI as a sequence of characters from the UCS normalized according to Unicode Normalization Form C (NFC, [UTR15]).

3.2. Parse the IRI into IRI components

Parse the IRI, either as a relative reference (no scheme) or using scheme specific processing (according to the scheme given); the result resulting in a set of parsed IRI components. (NOTE: FIX BEFORE RELEASE: INTENT IS THAT ALL IRI SCHEMES THAT USE GENERIC SYNTAX AND ALLOW NON-ASCII AUTHORITY CAN ONLY USE AUTHORITY FOR NAMES THAT FOLLOW PUNICODE.)

NOTE: The result of parsing into components will correspond result in a correspondence of subtrings of the IRI according to the part matched. For example, in [HTML5], the protocol components of interest are SCHEME (scheme), HOST (ireg-name), PORT (port), the PATH (ipath after the initial "/"), QUERY (iquery), FRAGMENT (ifragment), and AUTHORITY (iauthority).

Subsequent processing rules are sometimes used to define other syntactic components. For example, [HTML5] defines APIs for IRI

processing; in these APIs:

HOSTSPECIFIC the substring that follows the substring matched by the iauthority production, or the whole string if the iauthority production wasn't matched.

HOSTPORT if there is a scheme component and a port component and the port given by the port component is different than the default port defined for the protocol given by the scheme component, then HOSTPORT is the substring that starts with the substring matched by the host production and ends with the substring matched by the port production, and includes the colon in between the two. Otherwise, it is the same as the host component.

3.3. General percent-encoding of IRI components

For most IRI components, it is possible to map the IRI component to an equivalent URI component by percent-encoding those characters not allowed in URIs. Previous processing steps will have removed some characters, and the interpretation of reserved characters will have already been done (with the syntactic reserved characters outside of the IRI component). This mapping is defined for all sequences of Unicode characters, whether or not they are valid for the component in question.

For each character which is not allowed in a valid URI (NOTE: WHAT IS THE RIGHT REFERENCE HERE), apply the following steps.

Convert to UTF-8 Convert the character to a sequence of one or more octets using UTF-8 [RFC3629].

Percent encode Convert each octet of this sequence to %HH, where HH is the hexadecimal notation of the octet value. The hexadecimal notation SHOULD use uppercase letters. (This is the general URI percent-encoding mechanism in Section 2.1 of [RFC3986].)

Note that the mapping is an identity transformation for parsed URI components of valid URIs, and is idempotent: applying the mapping a second time will not change anything.

3.4. Mapping ireg-name

Schemes that allow non-ASCII based characters in the reg-name (ireg-name) position MUST convert the ireg-name component of an IRI as follows:

Replace the ireg-name part of the IRI by the part converted using the ToASCII operation specified in Section 4.1 of [RFC3490] on each dot-

separated label, and by using U+002E (FULL STOP) as a label separator, with the flag UseSTD3ASCIIRules set to FALSE, and with the flag AllowUnassigned set to FALSE. The ToASCII operation may fail, but this would mean that the IRI cannot be resolved. In such cases, if the domain name conversion fails, then the entire IRI conversion fails. Processors that have no mechanism for signalling a failure MAY instead substitute an otherwise invalid host name, although such processing SHOULD be avoided.

For example, the IRI
"http://résumé.example.org"
MAY be converted to
"http://xn--rsum-bad.example.org"
; conversion to percent-encoded form, e.g.,
"http://r%C3%A9sum%C3%A9.example.org", MUST NOT be performed.

Note: Domain Names may appear in parts of an IRI other than the ireg-name part. It is the responsibility of scheme-specific implementations (if the Internationalized Domain Name is part of the scheme syntax) or of server-side implementations (if the Internationalized Domain Name is part of 'iquery') to apply the necessary conversions at the appropriate point. Example: Trying to validate the Web page at http://résumé.example.org would lead to an IRI of http://validator.w3.org/check?uri=http%3A%2F%2Frésumé.example.org, which would convert to a URI of http://validator.w3.org/check?uri=http%3A%2F%2Fr%C3%A9sum%C3%A9.example.org. The server-side implementation is responsible for making the necessary conversions to be able to retrieve the Web page.

Note: In this process, characters allowed in URI references and existing percent-encoded sequences are not encoded further. (This mapping is similar to, but different from, the encoding applied when arbitrary content is included in some part of a URI.) For example, an IRI of
"http://www.example.org/red%09rosé#red" (in XML notation) is converted to
"http://www.example.org/red%09ros%C3%A9#red", not to something like
"http%3A%2F%2Fwww.example.org%2Fred%2509ros%C3%A9%23red".
((DESIGN QUESTION: What about e.g. http://r%C3%A9sum%C3%A9.example.org in an IRI? Will that get converted to punycode, or not?))

3.5. Mapping query components

((NOTE: SEE ISSUES LIST)) For compatibility with existing deployed HTTP infrastructure, the following special case applies for schemes "http" and "https" and IRIs whose origin has a document charset other than one which is UCS-based (e.g., UTF-8 or UTF-16). In such a case, the "query" component of an IRI is mapped into a URI by using the document charset rather than UTF-8 as the binary representation before pct-encoding. This mapping is not applied for any other scheme or component.

3.6. Mapping IRIs to URIs

The canonical mapping from a IRI to URI is defined by applying the mapping above (from IRI to URI components) and then reassembling a URI from the parsed URI components using the original punctuation that delimited the IRI components.

3.7. Converting URIs to IRIs

In some situations, for presentation and further processing, it is desirable to convert a URI into an equivalent IRI in which natural characters are represented directly rather than percent encoded. Of course, every URI is already an IRI in its own right without any conversion, and in general there This section gives one such procedure for this conversion.

The conversion described in this section, if given a valid URI, will result in an IRI that maps back to the URI used as an input for the conversion (except for potential case differences in percent-encoding and for potential percent-encoded unreserved characters). However, the IRI resulting from this conversion may differ from the original IRI (if there ever was one).

URI-to-IRI conversion removes percent-encodings, but not all percent-encodings can be eliminated. There are several reasons for this:

1. Some percent-encodings are necessary to distinguish percent-encoded and unencoded uses of reserved characters.

2. Some percent-encodings cannot be interpreted as sequences of UTF-8 octets.

   (Note: The octet patterns of UTF-8 are highly regular. Therefore, there is a very high probability, but no guarantee, that percent-encodings that can be interpreted as sequences of UTF-8 octets actually originated from UTF-8. For a detailed discussion, see [Duerst97].)

3. The conversion may result in a character that is not appropriate in an IRI. See Section 2.2, Section 4.1, and Section 6.1 for further details.

4. IRI to URI conversion has different rules for dealing with domain names and query parameters.

Conversion from a URI to an IRI MAY be done by using the following steps:

1. Represent the URI as a sequence of octets in US-ASCII.

2. Convert all percent-encodings ("%" followed by two hexadecimal digits) to the corresponding octets, except those corresponding to "%", characters in "reserved", and characters in US-ASCII not allowed in URIs.

3. Re-percent-encode any octet produced in step 2 that is not part of a strictly legal UTF-8 octet sequence.

4. Re-percent-encode all octets produced in step 3 that in UTF-8 represent characters that are not appropriate according to Section 2.2, Section 4.1, and Section 6.1.

5. Interpret the resulting octet sequence as a sequence of characters encoded in UTF-8.

6. URIs known to contain domain names in the reg-name component SHOULD convert punycode-encoded domain name labels to the corresponding characters using the ToUnicode procedure.

This procedure will convert as many percent-encoded characters as possible to characters in an IRI. Because there are some choices when step 4 is applied (see Section 6.1), results may vary.

Conversions from URIs to IRIs MUST NOT use any character encoding other than UTF-8 in steps 3 and 4, even if it might be possible to guess from the context that another character encoding than UTF-8 was used in the URI. For example, the URI "http://www.example.org/r%E9sum%E9.html" might with some guessing be interpreted to contain two e-acute characters encoded as iso-8859-1. It must not be converted to an IRI containing these e-acute characters. Otherwise, in the future the IRI will be mapped to "http://www.example.org/r%C3%A9sum%C3%A9.html", which is a different URI from "http://www.example.org/r%E9sum%E9.html".

3.7.1. Examples

This section shows various examples of converting URIs to IRIs. Each example shows the result after each of the steps 1 through 6 is applied. XML Notation is used for the final result. Octets are denoted by "<" followed by two hexadecimal digits followed by ">".

The following example contains the sequence "%C3%BC", which is a strictly legal UTF-8 sequence, and which is converted into the actual character U+00FC, LATIN SMALL LETTER U WITH DIAERESIS (also known as u-umlaut).

1. http://www.example.org/D%C3%BCrst

2. http://www.example.org/D<c3><bc>rst

3. http://www.example.org/D<c3><bc>rst

4. http://www.example.org/D<c3><bc>rst

5. http://www.example.org/Dürst

6. http://www.example.org/Dürst

The following example contains the sequence "%FC", which might represent U+00FC, LATIN SMALL LETTER U WITH DIAERESIS, in the iso-8859-1 character encoding. (It might represent other characters in other character encodings. For example, the octet <fc> in iso-8859-5 represents U+045C, CYRILLIC SMALL LETTER KJE.) Because <fc> is not part of a strictly legal UTF-8 sequence, it is re-percent-encoded in step 3.

1. http://www.example.org/D%FCrst

2. http://www.example.org/D<fc>rst

3. http://www.example.org/D%FCrst

4. http://www.example.org/D%FCrst

5. http://www.example.org/D%FCrst

6. http://www.example.org/D%FCrst

The following example contains "%e2%80%ae", which is the percent-encoded
UTF-8 character encoding of U+202E, RIGHT-TO-LEFT OVERRIDE.
Section 4.1 forbids the direct use of this character in an IRI.

Therefore, the corresponding octets are re-percent-encoded in step 4. This example shows that the case (upper- or lowercase) of letters used in percent-encodings may not be preserved. The example also contains a punycode-encoded domain name label (xn--99zt52a), which is not converted.

1. http://xn--99zt52a.example.org/%e2%80%ae

2. http://xn--99zt52a.example.org/<e2><80><ae>

3. http://xn--99zt52a.example.org/<e2><80><ae>

4. http://xn--99zt52a.example.org/%E2%80%AE

5. http://xn--99zt52a.example.org/%E2%80%AE

6. http://納豆.example.org/%E2%80%AE

Note that the label "xn--99zt52a" is converted to U+7D0D U+8C46 (Japanese Natto). ((EDITOR NOTE: There is some inconsistency in this note.))

4. Bidirectional IRIs for Right-to-Left Languages

Some UCS characters, such as those used in the Arabic and Hebrew scripts, have an inherent right-to-left (rtl) writing direction. IRIs containing these characters (called bidirectional IRIs or Bidi IRIs) require additional attention because of the non-trivial relation between logical representation (used for digital representation and for reading/spelling) and visual representation (used for display/printing).

Because of the complex interaction between the logical representation, the visual representation, and the syntax of a Bidi IRI, a balance is needed between various requirements. The main requirements are

1. user-predictable conversion between visual and logical representation;

2. the ability to include a wide range of characters in various parts of the IRI; and

3. minor or no changes or restrictions for implementations.

4.1. Logical Storage and Visual Presentation

When stored or transmitted in digital representation, bidirectional IRIs MUST be in full logical order and MUST conform to the IRI syntax rules (which includes the rules relevant to their scheme). This ensures that bidirectional IRIs can be processed in the same way as other IRIs.

Bidirectional IRIs MUST be rendered by using the Unicode Bidirectional Algorithm [UNIV4], [UNI9]. Bidirectional IRIs MUST be rendered in the same way as they would be if they were in a left-to-right embedding; i.e., as if they were preceded by U+202A, LEFT-TO-RIGHT EMBEDDING (LRE), and followed by U+202C, POP DIRECTIONAL FORMATTING (PDF). Setting the embedding direction can also be done in a higher-level protocol (e.g., the dir='ltr' attribute in HTML).

There is no requirement to use the above embedding if the display is still the same without the embedding. For example, a bidirectional IRI in a text with left-to-right base directionality (such as used for English or Cyrillic) that is preceded and followed by whitespace and strong left-to-right characters does not need an embedding. Also, a bidirectional relative IRI reference that only contains strong right-to-left characters and weak characters and that starts and ends with a strong right-to-left character and appears in a text with right-to-left base directionality (such as used for Arabic or Hebrew) and is preceded and followed by whitespace and strong characters does not need an embedding.

In some other cases, using U+200E, LEFT-TO-RIGHT MARK (LRM), may be sufficient to force the correct display behavior. However, the details of the Unicode Bidirectional algorithm are not always easy to understand. Implementers are strongly advised to err on the side of caution and to use embedding in all cases where they are not completely sure that the display behavior is unaffected without the embedding.

The Unicode Bidirectional Algorithm ([UNI9], section 4.3) permits higher-level protocols to influence bidirectional rendering. Such changes by higher-level protocols MUST NOT be used if they change the rendering of IRIs.

The bidirectional formatting characters that may be used before or after the IRI to ensure correct display are not themselves part of the IRI. IRIs MUST NOT contain bidirectional formatting characters (LRM, RLM, LRE, RLE, LRO, RLO, and PDF). They affect the visual rendering of the IRI but do not appear themselves. It would therefore not be possible to input an IRI with such characters correctly.

4.2. Bidi IRI Structure

The Unicode Bidirectional Algorithm is designed mainly for running text. To make sure that it does not affect the rendering of bidirectional IRIs too much, some restrictions on bidirectional IRIs are necessary. These restrictions are given in terms of delimiters (structural characters, mostly punctuation such as "@", ".", ":", and "/") and components (usually consisting mostly of letters and digits).

The following syntax rules from Section 2.2 correspond to components for the purpose of Bidi behavior: iuserinfo, ireg-name, isegment, isegment-nz, isegment-nz-nc, ireg-name, iquery, and ifragment.

Specifications that define the syntax of any of the above components MAY divide them further and define smaller parts to be components according to this document. As an example, the restrictions of [RFC3490] on bidirectional domain names correspond to treating each label of a domain name as a component for schemes with ireg-name as a domain name. Even where the components are not defined formally, it may be helpful to think about some syntax in terms of components and to apply the relevant restrictions. For example, for the usual name/value syntax in query parts, it is convenient to treat each name and each value as a component. As another example, the extensions in a resource name can be treated as separate components.

For each component, the following restrictions apply:

1. A component SHOULD NOT use both right-to-left and left-to-right characters.

2. A component using right-to-left characters SHOULD start and end with right-to-left characters.

The above restrictions are given as "SHOULD"s, rather than as "MUST"s. For IRIs that are never presented visually, they are not relevant. However, for IRIs in general, they are very important to ensure consistent conversion between visual presentation and logical representation, in both directions.

Note: In some components, the above restrictions may actually be strictly enforced. For example, [RFC3490] requires that these restrictions apply to the labels of a host name for those schemes where ireg-name is a host name. In some other components (for example, path components) following these restrictions may not be too difficult. For other components, such as parts of the query part, it may be very difficult to enforce the restrictions because the values of query parameters may be arbitrary character

sequences.

If the above restrictions cannot be satisfied otherwise, the affected component can always be mapped to URI notation as described in Section 3.3. Please note that the whole component has to be mapped (see also Example 9 below).

4.3. Input of Bidi IRIs

Bidi input methods MUST generate Bidi IRIs in logical order while rendering them according to Section 4.1. During input, rendering SHOULD be updated after every new character is input to avoid end-user confusion.

4.4. Examples

This section gives examples of bidirectional IRIs, in Bidi Notation. It shows legal IRIs with the relationship between logical and visual representation and explains how certain phenomena in this relationship may look strange to somebody not familiar with bidirectional behavior, but familiar to users of Arabic and Hebrew. It also shows what happens if the restrictions given in Section 4.2 are not followed. The examples below can be seen at [BidiEx], in Arabic, Hebrew, and Bidi Notation variants.

To read the bidi text in the examples, read the visual representation from left to right until you encounter a block of rtl text. Read the rtl block (including slashes and other special characters) from right to left, then continue at the next unread ltr character.

Example 1: A single component with rtl characters is inverted:
Logical representation: "http://ab.CDEFGH.ij/kl/mn/op.html"
Visual representation: "http://ab.HGFEDC.ij/kl/mn/op.html"
Components can be read one by one, and each component can be read in its natural direction.

Example 2: More than one consecutive component with rtl characters is inverted as a whole:
Logical representation: "http://ab.CDE.FGH/ij/kl/mn/op.html"
Visual representation: "http://ab.HGF.EDC/ij/kl/mn/op.html"
A sequence of rtl components is read rtl, in the same way as a sequence of rtl words is read rtl in a bidi text.

Example 3: All components of an IRI (except for the scheme) are rtl. All rtl components are inverted overall:
Logical representation: "http://AB.CD.EF/GH/IJ/KL?MN=OP;QR=ST#UV"
Visual representation: "http://VU#TS=RQ;PO=NM?LK/JI/HG/FE.DC.BA"
The whole IRI (except the scheme) is read rtl. Delimiters between

rtl components stay between the respective components; delimiters between ltr and rtl components don't move.

Example 4: Each of several sequences of rtl components is inverted on its own:
Logical representation: "http://AB.CD.ef/gh/IJ/KL.html"
Visual representation: "http://DC.BA.ef/gh/LK/JI.html"
Each sequence of rtl components is read rtl, in the same way as each sequence of rtl words in an ltr text is read rtl.

Example 5: Example 2, applied to components of different kinds:
Logical representation: "http://ab.cd.EF/GH/ij/kl.html"
Visual representation: "http://ab.cd.HG/FE/ij/kl.html"
The inversion of the domain name label and the path component may be unexpected, but it is consistent with other bidi behavior. For reassurance that the domain component really is "ab.cd.EF", it may be helpful to read aloud the visual representation following the bidi algorithm. After "http://ab.cd." one reads the RTL block "E-F-slash-G-H", which corresponds to the logical representation.

Example 6: Same as Example 5, with more rtl components:
Logical representation: "http://ab.CD.EF/GH/IJ/kl.html"
Visual representation: "http://ab.JI/HG/FE.DC/kl.html"
The inversion of the domain name labels and the path components may be easier to identify because the delimiters also move.

Example 7: A single rtl component includes digits:
Logical representation: "http://ab.CDE123FGH.ij/kl/mn/op.html"
Visual representation: "http://ab.HGF123EDC.ij/kl/mn/op.html"
Numbers are written ltr in all cases but are treated as an additional embedding inside a run of rtl characters. This is completely consistent with usual bidirectional text.

Example 8 (not allowed): Numbers are at the start or end of an rtl component:
Logical representation: "http://ab.cd.ef/GH1/2IJ/KL.html"
Visual representation: "http://ab.cd.ef/LK/JI1/2HG.html"
The sequence "1/2" is interpreted by the bidi algorithm as a fraction, fragmenting the components and leading to confusion. There are other characters that are interpreted in a special way close to numbers; in particular, "+", "-", "#", "$", "%", ",", ".", and ":".

Example 9 (not allowed): The numbers in the previous example are percent-encoded:
Logical representation: "http://ab.cd.ef/GH%31/%32IJ/KL.html",
Visual representation: "http://ab.cd.ef/LK/JI%32/%31HG.html"

Example 10 (allowed but not recommended):

Logical representation: "http://ab.CDEFGH.123/kl/mn/op.html"
Visual representation: "http://ab.123.HGFEDC/kl/mn/op.html"
Components consisting of only numbers are allowed (it would be rather difficult to prohibit them), but these may interact with adjacent RTL components in ways that are not easy to predict.

Example 11 (allowed but not recommended):
Logical representation: "http://ab.CDEFGH.123ij/kl/mn/op.html"
Visual representation: "http://ab.123.HGFEDCij/kl/mn/op.html"
Components consisting of numbers and left-to-right characters are allowed, but these may interact with adjacent RTL components in ways that are not easy to predict.

5. Normalization and Comparison

Note: The structure and much of the material for this section is taken from section 6 of [RFC3986]; the differences are due to the specifics of IRIs.

One of the most common operations on IRIs is simple comparison: Determining whether two IRIs are equivalent, without using the IRIs to access their respective resource(s). A comparison is performed whenever a response cache is accessed, a browser checks its history to color a link, or an XML parser processes tags within a namespace. Extensive normalization prior to comparison of IRIs may be used by spiders and indexing engines to prune a search space or reduce duplication of request actions and response storage.

IRI comparison is performed for some particular purpose. Protocols or implementations that compare IRIs for different purposes will often be subject to differing design trade-offs in regards to how much effort should be spent in reducing aliased identifiers. This section describes various methods that may be used to compare IRIs, the trade-offs between them, and the types of applications that might use them.

5.1. Equivalence

Because IRIs exist to identify resources, presumably they should be considered equivalent when they identify the same resource. However, this definition of equivalence is not of much practical use, as there is no way for an implementation to compare two resources to determine if they are "the same" unless it has full knowledge or control of them. For this reason, determination of equivalence or difference of IRIs is based on string comparison, perhaps augmented by reference to additional rules provided by URI scheme definitions. We use the terms "different" and "equivalent" to describe the possible outcomes

of such comparisons, but there are many application-dependent versions of equivalence.

Even when it is possible to determine that two IRIs are equivalent, IRI comparison is not sufficient to determine whether two IRIs identify different resources. For example, an owner of two different domain names could decide to serve the same resource from both, resulting in two different IRIs. Therefore, comparison methods are designed to minimize false negatives while strictly avoiding false positives.

In testing for equivalence, applications should not directly compare relative references; the references should be converted to their respective target IRIs before comparison. When IRIs are compared to select (or avoid) a network action, such as retrieval of a representation, fragment components (if any) should be excluded from the comparison.

Applications using IRIs as identity tokens with no relationship to a protocol MUST use the Simple String Comparison (see Section 5.3.1). All other applications MUST select one of the comparison practices from the Comparison Ladder (see Section 5.3.

5.2. Preparation for Comparison

Any kind of IRI comparison REQUIRES that any additional contextual processing is first performed, including undoing higher-level escapings or encodings in the protocol or format that carries an IRI. This preprocessing is usually done when the protocol or format is parsed.

Examples of contextual preprocessing steps are described in Section 7.

Examples of such escapings or encodings are entities and numeric character references in [HTML4] and [XML1]. As an example, "http://example.org/rosé" (in HTML), "http://example.org/rosé" (in HTML or XML), and "http://example.org/rosé" (in HTML or XML) are all resolved into what is denoted in this document (see Section 1.4) as "http://example.org/rosé" (the "é" here standing for the actual e-acute character, to compensate for the fact that this document cannot contain non-ASCII characters).

Similar considerations apply to encodings such as Transfer Codings in HTTP (see [RFC2616]) and Content Transfer Encodings in MIME ([RFC2045]), although in these cases, the encoding is based not on characters but on octets, and additional care is required to make

sure that characters, and not just arbitrary octets, are compared (see Section 5.3.1).

5.3. Comparison Ladder

In practice, a variety of methods are used to test IRI equivalence. These methods fall into a range distinguished by the amount of processing required and the degree to which the probability of false negatives is reduced. As noted above, false negatives cannot be eliminated. In practice, their probability can be reduced, but this reduction requires more processing and is not cost-effective for all applications.

If this range of comparison practices is considered as a ladder, the following discussion will climb the ladder, starting with practices that are cheap but have a relatively higher chance of producing false negatives, and proceeding to those that have higher computational cost and lower risk of false negatives.

5.3.1. Simple String Comparison

If two IRIs, when considered as character strings, are identical, then it is safe to conclude that they are equivalent. This type of equivalence test has very low computational cost and is in wide use in a variety of applications, particularly in the domain of parsing. It is also used when a definitive answer to the question of IRI equivalence is needed that is independent of the scheme used and that can be calculated quickly and without accessing a network. An example of such a case is XML Namespaces ([XMLNamespace]).

Testing strings for equivalence requires some basic precautions. This procedure is often referred to as "bit-for-bit" or "byte-for-byte" comparison, which is potentially misleading. Testing strings for equality is normally based on pair comparison of the characters that make up the strings, starting from the first and proceeding until both strings are exhausted and all characters are found to be equal, until a pair of characters compares unequal, or until one of the strings is exhausted before the other.

This character comparison requires that each pair of characters be put in comparable encoding form. For example, should one IRI be stored in a byte array in UTF-8 encoding form and the second in a UTF-16 encoding form, bit-for-bit comparisons applied naively will produce errors. It is better to speak of equality on a character-for-character rather than on a byte-for-byte or bit-for-bit basis. In practical terms, character-by-character comparisons should be done codepoint by codepoint after conversion to a common character encoding form. When comparing character by character, the comparison

function MUST NOT map IRIs to URIs, because such a mapping would create additional spurious equivalences. It follows that an IRI SHOULD NOT be modified when being transported if there is any chance that this IRI might be used in a context that uses Simple String Comparison.

False negatives are caused by the production and use of IRI aliases. Unnecessary aliases can be reduced, regardless of the comparison method, by consistently providing IRI references in an already normalized form (i.e., a form identical to what would be produced after normalization is applied, as described below). Protocols and data formats often limit some IRI comparisons to simple string comparison, based on the theory that people and implementations will, in their own best interest, be consistent in providing IRI references, or at least be consistent enough to negate any efficiency that might be obtained from further normalization.

5.3.2. Syntax-Based Normalization

Implementations may use logic based on the definitions provided by this specification to reduce the probability of false negatives. This processing is moderately higher in cost than character-for-character string comparison. For example, an application using this approach could reasonably consider the following two IRIs equivalent:

```
example://a/b/c/%7Bfoo%7D/ros&#xE9;
eXAMPLE://a/./b/../b/%63/%7bfoo%7d/ros%C3%A9
```

Web user agents, such as browsers, typically apply this type of IRI normalization when determining whether a cached response is available. Syntax-based normalization includes such techniques as case normalization, character normalization, percent-encoding normalization, and removal of dot-segments.

5.3.2.1. Case Normalization

For all IRIs, the hexadecimal digits within a percent-encoding triplet (e.g., "%3a" versus "%3A") are case-insensitive and therefore should be normalized to use uppercase letters for the digits A-F.

When an IRI uses components of the generic syntax, the component syntax equivalence rules always apply; namely, that the scheme and US-ASCII only host are case insensitive and therefore should be normalized to lowercase. For example, the URI "HTTP://www.EXAMPLE.com/" is equivalent to "http://www.example.com/". Case equivalence for non-ASCII characters in IRI components that are IDNs are discussed in Section 5.3.3. The other generic syntax components are assumed to be case sensitive unless specifically

defined otherwise by the scheme.

Creating schemes that allow case-insensitive syntax components containing non-ASCII characters should be avoided. Case normalization of non-ASCII characters can be culturally dependent and is always a complex operation. The only exception concerns non-ASCII host names for which the character normalization includes a mapping step derived from case folding.

5.3.2.2. Character Normalization

The Unicode Standard [UNIV4] defines various equivalences between sequences of characters for various purposes. Unicode Standard Annex #15 [UTR15] defines various Normalization Forms for these equivalences, in particular Normalization Form C (NFC, Canonical Decomposition, followed by Canonical Composition) and Normalization Form KC (NFKC, Compatibility Decomposition, followed by Canonical Composition).

IRIs already in Unicode MUST NOT be normalized before parsing or interpreting. In many non-Unicode character encodings, some text cannot be represented directly. For example, the word "Vietnam" is natively written "Việt Nam" (containing a LATIN SMALL LETTER E WITH CIRCUMFLEX AND DOT BELOW) in NFC, but a direct transcoding from the windows-1258 character encoding leads to "Việt Nam" (containing a LATIN SMALL LETTER E WITH CIRCUMFLEX followed by a COMBINING DOT BELOW). Direct transcoding of other 8-bit encodings of Vietnamese may lead to other representations.

Equivalence of IRIs MUST rely on the assumption that IRIs are appropriately pre-character-normalized rather than apply character normalization when comparing two IRIs. The exceptions are conversion from a non-digital form, and conversion from a non-UCS-based character encoding to a UCS-based character encoding. In these cases, NFC or a normalizing transcoder using NFC MUST be used for interoperability. To avoid false negatives and problems with transcoding, IRIs SHOULD be created by using NFC. Using NFKC may avoid even more problems; for example, by choosing half-width Latin letters instead of full-width ones, and full-width instead of half-width Katakana.

As an example, "http://www.example.org/résumé.html" (in XML Notation) is in NFC. On the other hand, "http://www.example.org/résumé.html" is not in NFC.

The former uses precombined e-acute characters, and the latter uses "e" characters followed by combining acute accents. Both usages are defined as canonically equivalent in [UNIV4].

Note: Because it is unknown how a particular sequence of characters is being treated with respect to character normalization, it would be inappropriate to allow third parties to normalize an IRI arbitrarily. This does not contradict the recommendation that when a resource is created, its IRI should be as character normalized as possible (i.e., NFC or even NFKC). This is similar to the uppercase/lowercase problems. Some parts of a URI are case insensitive (for example, the domain name). For others, it is unclear whether they are case sensitive, case insensitive, or something in between (e.g., case sensitive, but with a multiple choice selection if the wrong case is used, instead of a direct negative result). The best recipe is that the creator use a reasonable capitalization and, when transferring the URI, capitalization never be changed.

Various IRI schemes may allow the usage of Internationalized Domain Names (IDN) [RFC3490] either in the ireg-name part or elsewhere. Character Normalization also applies to IDNs, as discussed in Section 5.3.3.

5.3.2.3. Percent-Encoding Normalization

The percent-encoding mechanism (Section 2.1 of [RFC3986]) is a frequent source of variance among otherwise identical IRIs. In addition to the case normalization issue noted above, some IRI producers percent-encode octets that do not require percent-encoding, resulting in IRIs that are equivalent to their nonencoded counterparts. These IRIs should be normalized by decoding any percent-encoded octet sequence that corresponds to an unreserved character, as described in section 2.3 of [RFC3986].

For actual resolution, differences in percent-encoding (except for the percent-encoding of reserved characters) MUST always result in the same resource. For example, "http://example.org/~user", "http://example.org/%7euser", and "http://example.org/%7Euser", must resolve to the same resource.

If this kind of equivalence is to be tested, the percent-encoding of both IRIs to be compared has to be aligned; for example, by converting both IRIs to URIs (see Section 3.1), eliminating escape differences in the resulting URIs, and making sure that the case of the hexadecimal characters in the percent-encoding is always the same (preferably upper case). If the IRI is to be passed to another application or used further in some other way, its original form MUST be preserved. The conversion described here should be performed only for local comparison.

5.3.2.4. Path Segment Normalization

The complete path segments "." and ".." are intended only for use within relative references (Section 4.1 of [RFC3986]) and are removed as part of the reference resolution process (Section 5.2 of [RFC3986]). However, some implementations may incorrectly assume that reference resolution is not necessary when the reference is already an IRI, and thus fail to remove dot-segments when they occur in non-relative paths. IRI normalizers should remove dot-segments by applying the remove_dot_segments algorithm to the path, as described in Section 5.2.4 of [RFC3986].

5.3.3. Scheme-Based Normalization

The syntax and semantics of IRIs vary from scheme to scheme, as described by the defining specification for each scheme. Implementations may use scheme-specific rules, at further processing cost, to reduce the probability of false negatives. For example, because the "http" scheme makes use of an authority component, has a default port of "80", and defines an empty path to be equivalent to "/", the following four IRIs are equivalent:

```
http://example.com
http://example.com/
http://example.com:/
http://example.com:80/
```

In general, an IRI that uses the generic syntax for authority with an empty path should be normalized to a path of "/". Likewise, an explicit ":port", for which the port is empty or the default for the scheme, is equivalent to one where the port and its ":" delimiter are elided and thus should be removed by scheme-based normalization. For example, the second IRI above is the normal form for the "http" scheme.

Another case where normalization varies by scheme is in the handling of an empty authority component or empty host subcomponent. For many scheme specifications, an empty authority or host is considered an error; for others, it is considered equivalent to "localhost" or the end-user's host. When a scheme defines a default for authority and an IRI reference to that default is desired, the reference should be normalized to an empty authority for the sake of uniformity, brevity, and internationalization. If, however, either the userinfo or port subcomponents are non-empty, then the host should be given explicitly even if it matches the default.

Normalization should not remove delimiters when their associated component is empty unless it is licensed to do so by the scheme

specification. For example, the IRI "http://example.com/?" cannot be assumed to be equivalent to any of the examples above. Likewise, the presence or absence of delimiters within a userinfo subcomponent is usually significant to its interpretation. The fragment component is not subject to any scheme-based normalization; thus, two IRIs that differ only by the suffix "#" are considered different regardless of the scheme.

((NOTE: THIS NEEDS TO BE UPDATED TO DEAL WITH IDNA8)) Some IRI schemes may allow the usage of Internationalized Domain Names (IDN) [RFC3490] either in their ireg-name part or elsewhere. When in use in IRIs, those names SHOULD be validated by using the ToASCII operation defined in [RFC3490], with the flags "UseSTD3ASCIIRules" and "AllowUnassigned". An IRI containing an invalid IDN cannot successfully be resolved. Validated IDN components of IRIs SHOULD be character normalized by using the Nameprep process [RFC3491]; however, for legibility purposes, they SHOULD NOT be converted into ASCII Compatible Encoding (ACE).

Scheme-based normalization may also consider IDN components and their conversions to punycode as equivalent. As an example, "http://résumé.example.org" may be considered equivalent to "http://xn--rsum-bpad.example.org".

Other scheme-specific normalizations are possible.

5.3.4. Protocol-Based Normalization

Substantial effort to reduce the incidence of false negatives is often cost-effective for web spiders. Consequently, they implement even more aggressive techniques in IRI comparison. For example, if they observe that an IRI such as

   http://example.com/data

redirects to an IRI differing only in the trailing slash

   http://example.com/data/

they will likely regard the two as equivalent in the future. This kind of technique is only appropriate when equivalence is clearly indicated by both the result of accessing the resources and the common conventions of their scheme's dereference algorithm (in this case, use of redirection by HTTP origin servers to avoid problems with relative references).

6. Use of IRIs

6.1. Limitations on UCS Characters Allowed in IRIs

This section discusses limitations on characters and character sequences usable for IRIs beyond those given in Section 2.2 and Section 4.1. The considerations in this section are relevant when IRIs are created and when URIs are converted to IRIs.

a. The repertoire of characters allowed in each IRI component is limited by the definition of that component. For example, the definition of the scheme component does not allow characters beyond US-ASCII.

   (Note: In accordance with URI practice, generic IRI software cannot and should not check for such limitations.)

b. The UCS contains many areas of characters for which there are strong visual look-alikes. Because of the likelihood of transcription errors, these also should be avoided. This includes the full-width equivalents of Latin characters, half-width Katakana characters for Japanese, and many others. It also includes many look-alikes of "space", "delims", and "unwise", characters excluded in [RFC3491].

Additional information is available from [UNIXML]. [UNIXML] is written in the context of running text rather than in that of identifiers. Nevertheless, it discusses many of the categories of characters not appropriate for IRIs.

6.2. Software Interfaces and Protocols

Although an IRI is defined as a sequence of characters, software interfaces for URIs typically function on sequences of octets or other kinds of code units. Thus, software interfaces and protocols MUST define which character encoding is used.

Intermediate software interfaces between IRI-capable components and URI-only components MUST map the IRIs per Section 3.6, when transferring from IRI-capable to URI-only components. This mapping SHOULD be applied as late as possible. It SHOULD NOT be applied between components that are known to be able to handle IRIs.

6.3. Format of URIs and IRIs in Documents and Protocols

Document formats that transport URIs may have to be upgraded to allow the transport of IRIs. In cases where the document as a whole has a native character encoding, IRIs MUST also be encoded in this

character encoding and converted accordingly by a parser or interpreter. IRI characters not expressible in the native character encoding SHOULD be escaped by using the escaping conventions of the document format if such conventions are available. Alternatively, they MAY be percent-encoded according to Section 3.6. For example, in HTML or XML, numeric character references SHOULD be used. If a document as a whole has a native character encoding and that character encoding is not UTF-8, then IRIs MUST NOT be placed into the document in the UTF-8 character encoding.

((UPDATE THIS NOTE)) Note: Some formats already accommodate IRIs, although they use different terminology. HTML 4.0 [HTML4] defines the conversion from IRIs to URIs as error-avoiding behavior. XML 1.0 [XML1], XLink [XLink], XML Schema [XMLSchema], and specifications based upon them allow IRIs. Also, it is expected that all relevant new W3C formats and protocols will be required to handle IRIs [CharMod].

6.4. Use of UTF-8 for Encoding Original Characters

This section discusses details and gives examples for point c) in Section 1.2. To be able to use IRIs, the URI corresponding to the IRI in question has to encode original characters into octets by using UTF-8. This can be specified for all URIs of a URI scheme or can apply to individual URIs for schemes that do not specify how to encode original characters. It can apply to the whole URI, or only to some part. For background information on encoding characters into URIs, see also Section 2.5 of [RFC3986].

For new URI schemes, using UTF-8 is recommended in [RFC4395]. Examples where UTF-8 is already used are the URN syntax [RFC2141], IMAP URLs [RFC2192], and POP URLs [RFC2384]. On the other hand, because the HTTP URI scheme does not specify how to encode original characters, only some HTTP URLs can have corresponding but different IRIs.

For example, for a document with a URI of "http://www.example.org/r%C3%A9sum%C3%A9.html", it is possible to construct a corresponding IRI (in XML notation, see Section 1.4): "http://www.example.org/résumé.html" ("é" stands for the e-acute character, and "%C3%A9" is the UTF-8 encoded and percent-encoded representation of that character). On the other hand, for a document with a URI of "http://www.example.org/r%E9sum%E9.html", the percent-encoding octets cannot be converted to actual characters in an IRI, as the percent-encoding is not based on UTF-8.

For most URI schemes, there is no need to upgrade their scheme definition in order for them to work with IRIs. The main case where

upgrading makes sense is when a scheme definition, or a particular component of a scheme, is strictly limited to the use of US-ASCII characters with no provision to include non-ASCII characters/octets via percent-encoding, or if a scheme definition currently uses highly scheme-specific provisions for the encoding of non-ASCII characters. An example of this is the mailto: scheme [RFC2368].

This specification updates the IANA registry of URI schemes to note their applicability to IRIs, see Section 9. All IRIs use URI schemes, and all URIs with URI schemes can be used as IRIs, even though in some cases only by using URIs directly as IRIs, without any conversion.

Scheme definitions can impose restrictions on the syntax of scheme-specific URIs; i.e., URIs that are admissible under the generic URI syntax [RFC3986] may not be admissible due to narrower syntactic constraints imposed by a URI scheme specification. URI scheme definitions cannot broaden the syntactic restrictions of the generic URI syntax; otherwise, it would be possible to generate URIs that satisfied the scheme-specific syntactic constraints without satisfying the syntactic constraints of the generic URI syntax. However, additional syntactic constraints imposed by URI scheme specifications are applicable to IRI, as the corresponding URI resulting from the mapping defined in Section 3.6 MUST be a valid URI under the syntactic restrictions of generic URI syntax and any narrower restrictions imposed by the corresponding URI scheme specification.

The requirement for the use of UTF-8 generally applies to all parts of a URI. However, it is possible that the capability of IRIs to represent a wide range of characters directly is used just in some parts of the IRI (or IRI reference). The other parts of the IRI may only contain US-ASCII characters, or they may not be based on UTF-8. They may be based on another character encoding, or they may directly encode raw binary data (see also [RFC2397]).

For example, it is possible to have a URI reference of "http://www.example.org/r%E9sum%E9.xml#r%C3%A9sum%C3%A9", where the document name is encoded in iso-8859-1 based on server settings, but where the fragment identifier is encoded in UTF-8 according to [XPointer]. The IRI corresponding to the above URI would be (in XML notation)
"http://www.example.org/r%E9sum%E9.xml#résumé".

Similar considerations apply to query parts. The functionality of IRIs (namely, to be able to include non-ASCII characters) can only be used if the query part is encoded in UTF-8.

6.5. Relative IRI References

Processing of relative IRI references against a base is handled straightforwardly; the algorithms of [RFC3986] can be applied directly, treating the characters additionally allowed in IRI references in the same way that unreserved characters are in URI references.

7. Liberal handling of otherwise invalid IRIs

(EDITOR NOTE: This Section may move to an appendix.) Some technical specifications and widely-deployed software have allowed additional variations and extensions of IRIs to be used in syntactic components. This section describes two widely-used preprocessing agreements. Other technical specifications may wish to reference a syntactic component which is "a valid IRI or a string that will map to a valid IRI after this preprocessing algorithm". These two variants are known as Legacy Extended IRI or LEIRI [LEIRI], and Web Address [HTML5]).

Future technical specifications SHOULD NOT allow conforming producers to produce, or conforming content to contain, such forms, as they are not interoperable with other IRI consuming software.

7.1. LEIRI processing

This section defines Legacy Extended IRIs (LEIRIs). The syntax of Legacy Extended IRIs is the same as that for IRIs, except that the ucschar production is replaced by the leiri-ucschar production:

```
leiri-ucschar  = " " / "<" / ">" / '"' / "{" / "}" / "|"
                 / "\" / "^" / "`" / %x0-1F / %x7F-D7FF
                 / %xE000-FFFD / %x10000-10FFFF
```

Among other extensions, processors based on this specification also did not enforce the restriction on bidirectional formatting characters in Section 4.1, and the iprivate production becomes redundant.

To convert a string allowed as a LEIRI to an IRI, each character allowed in leiri-ucschar but not in ucschar must be percent-encoded using Section 3.3.

7.2. Web Address processing

Many popular web browsers have taken the approach of being quite liberal in what is accepted as a "URL" or its relative forms. This

section describes their behavior in terms of a preprocessor which maps strings into the IRI space for subsequent parsing and interpretation as an IRI.

In some situations, it might be appropriate to describe the syntax that a liberal consumer implementation might accept as a "Web Address" or "Hypertext Reference" or "HREF". However, technical specifications SHOULD restrict the syntactic form allowed by compliant producers to the IRI or IRI reference syntax defined in this document even if they want to mandate this processing.

Summary:

- Leading and trailing whitespace is removed.
- Some additional characters are removed.
- Some additional characters are allowed and escaped (as with LEIRI).
- If interpreting an IRI as a URI, the pct-encoding of the query component of the parsed URI component depends on operational context.

Each string provided may have an associated charset (called the HREF-charset here); this defaults to UTF-8. For web browsers interpreting HTML, the document charset of a string is determined:

If the string came from a script (e.g. as an argument to a method) The HRef-charset is the script's charset.

If the string came from a DOM node (e.g. from an element) The node has a Document, and the HRef-charset is the Document's character encoding.

If the string had a HRef-charset defined when the string was created or defined The HRef-charset is as defined.

If the resulting HRef-charset is a unicode based character encoding (e.g., UTF-16), then use UTF-8 instead.

The syntax for Web Addresses is obtained by replacing the 'ucschar', pct-form, and path-sep rules with the href-ucschar, href-pct-form, and href-path-sep rules below. In addition, some characters are stripped.

```
  href-ucschar  = " " / "<" / ">" / '"' / "{" / "}" / "|"
                  / "\" / "^" / "`" / %x0-1F / %x7F-D7FF
                  / %xE000-FFFD / %x10000-10FFFF
  href-pct-form = pct-encoded | "%"
  href-path-sep = "/" | "\"
  href-strip    =
```

(NOTE: NEED TO FIX THESE SETS TO MATCH HTML5; NOT SURE ABOUT NEXT SENTENCE) browsers did not enforce the restriction on bidirectional formatting characters in Section 4.1, and the iprivate production becomes redundant.

'Web Address processing' requires the following additional preprocessing steps:

1. Leading and trailing instances of space (U+0020), CR (U+000A), LF (U+000D), and TAB (U+0009) characters are removed.

2. strip all characters in href-strip.

3. Percent-encode all characters in href-ucschar not in ucschar.

4. Replace occurrences of "%" not followed by two hexadecimal digits by "%25".

5. Convert backslashes ('\') matching href-path-sep to forward slashes ('/').

7.3. Characters not allowed in IRIs

This section provides a list of the groups of characters and code points that are allowed by LEIRI or HREF but are not allowed in IRIs or are allowed in IRIs only in the query part. For each group of characters, advice on the usage of these characters is also given, concentrating on the reasons for why they are excluded from IRI use.

Space (U+0020): Some formats and applications use space as a delimiter, e.g. for items in a list. Appendix C of [RFC3986] also mentions that white space may have to be added when displaying or printing long URIs; the same applies to long IRIs. This means that spaces can disappear, or can make the what is intended as a single IRI or IRI reference to be treated as two or more separate IRIs.

Delimiters "<" (U+003C), ">" (U+003E), and '"' (U+0022): Appendix C of [RFC3986] suggests the use of double-quotes ("http://example.com/") and angle brackets (<http://example.com/>) as delimiters for URIs in plain text. These conventions are often used, and also apply to IRIs. Using these characters in strings intended to be IRIs would result in the IRIs being cut off at the wrong place.

Unwise characters "\" (U+005C), "^" (U+005E), "`" (U+0060), "{" (U+007B), "|" (U+007C), and "}" (U+007D): These characters originally have been excluded from URIs because the respective codepoints are assigned to different graphic characters in some 7-bit or 8-bit encoding. Despite the move to Unicode, some of these characters are still occasionally displayed differently on some systems, e.g. U+005C may appear as a Japanese Yen symbol on some systems. Also, the fact that these characters are not used in URIs or IRIs has encouraged their use outside URIs or IRIs in contexts that may include URIs or IRIs. If a string with such a character were used as an IRI in such a context, it would likely be interpreted piecemeal.

The controls (C0 controls, DEL, and C1 controls, #x0 - #x1F #x7F - #x9F): There is generally no way to transmit these characters reliably as text outside of a charset encoding. Even when in encoded form, many software components silently filter out some of these characters, or may stop processing alltogether when encountering some of them. These characters may affect text display in subtle, unnoticable ways or in drastic, global, and irreversible ways depending on the hardware and software involved. The use of some of these characters would allow malicious users to manipulate the display of an IRI and its context in many situations.

Bidi formatting characters (U+200E, U+200F, U+202A-202E): These characters affect the display ordering of characters. If IRIs were allowed to contain these characters and the resulting visual display transcribed. they could not be converted back to electronic form (logical order) unambiguously. These characters, if allowed in IRIs, might allow malicious users to manipulate the display of IRI and its context.

Specials (U+FFF0-FFFD): These code points provide functionality beyond that useful in an IRI, for example byte order identification, annotation, and replacements for unknown characters and objects. Their use and interpretation in an IRI would serve no purpose and might lead to confusing display variations.

Private use code points (U+E000-F8FF, U+F0000-FFFFD, U+100000-10FFFD): Display and interpretation of these code points is by definition undefined without private agreement. Therefore, these code points are not suited for use on the Internet. They are not interoperable and may have unpredictable effects.

Tags (U+E0000-E0FFF): These characters provide a way to language tag in Unicode plain text. They are not appropriate for IRIs because language information in identifiers cannot reliably be input, transmitted (e.g. on a visual medium such as paper), or recognized.

Non-characters (U+FDD0-FDEF, U+1FFFE-1FFFF, U+2FFFE-2FFFF, U+3FFFE-3FFFF, U+4FFFE-4FFFF, U+5FFFE-5FFFF, U+6FFFE-6FFFF, U+7FFFE-7FFFF, U+8FFFE-8FFFF, U+9FFFE-9FFFF, U+AFFFE-AFFFF, U+BFFFE-BFFFF, U+CFFFE-CFFFF, U+DFFFE-DFFFF, U+EFFFE-EFFFF, U+FFFFE-FFFFF, U+10FFFE-10FFFF): These code points are defined as non-characters. Applications may use some of them internally, but are not prepared to interchange them.

LEIRI preprocessing disallowed some code points and code units:

Surrogate code units (D800-DFFF): These do not represent Unicode codepoints.

8. URI/IRI Processing Guidelines (Informative)

This informative section provides guidelines for supporting IRIs in the same software components and operations that currently process URIs: Software interfaces that handle URIs, software that allows users to enter URIs, software that creates or generates URIs, software that displays URIs, formats and protocols that transport URIs, and software that interprets URIs. These may all require modification before functioning properly with IRIs. The considerations in this section also apply to URI references and IRI references.

8.1. URI/IRI Software Interfaces

Software interfaces that handle URIs, such as URI-handling APIs and protocols transferring URIs, need interfaces and protocol elements that are designed to carry IRIs.

In case the current handling in an API or protocol is based on US-ASCII, UTF-8 is recommended as the character encoding for IRIs, as it is compatible with US-ASCII, is in accordance with the recommendations of [RFC2277], and makes converting to URIs easy. In

any case, the API or protocol definition must clearly define the character encoding to be used.

The transfer from URI-only to IRI-capable components requires no mapping, although the conversion described in Section 3.7 above may be performed. It is preferable not to perform this inverse conversion unless it is certain this can be done correctly.

8.2. URI/IRI Entry

Some components allow users to enter URIs into the system by typing or dictation, for example. This software must be updated to allow for IRI entry.

A person viewing a visual representation of an IRI (as a sequence of glyphs, in some order, in some visual display) or hearing an IRI will use an entry method for characters in the user's language to input the IRI. Depending on the script and the input method used, this may be a more or less complicated process.

The process of IRI entry must ensure, as much as possible, that the restrictions defined in Section 2.2 are met. This may be done by choosing appropriate input methods or variants/settings thereof, by appropriately converting the characters being input, by eliminating characters that cannot be converted, and/or by issuing a warning or error message to the user.

As an example of variant settings, input method editors for East Asian Languages usually allow the input of Latin letters and related characters in full-width or half-width versions. For IRI input, the input method editor should be set so that it produces half-width Latin letters and punctuation and full-width Katakana.

An input field primarily or solely used for the input of URIs/IRIs might allow the user to view an IRI as it is mapped to a URI. Places where the input of IRIs is frequent may provide the possibility for viewing an IRI as mapped to a URI. This will help users when some of the software they use does not yet accept IRIs.

An IRI input component interfacing to components that handle URIs, but not IRIs, must map the IRI to a URI before passing it to these components.

For the input of IRIs with right-to-left characters, please see Section 4.3.

8.3. URI/IRI Transfer between Applications

Many applications (for example, mail user agents) try to detect URIs appearing in plain text. For this, they use some heuristics based on URI syntax. They then allow the user to click on such URIs and retrieve the corresponding resource in an appropriate (usually scheme-dependent) application.

Such applications would need to be upgraded, in order to use the IRI syntax as a base for heuristics. In particular, a non-ASCII character should not be taken as the indication of the end of an IRI. Such applications also would need to make sure that they correctly convert the detected IRI from the character encoding of the document or application where the IRI appears, to the character encoding used by the system-wide IRI invocation mechanism, or to a URI (according to Section 3.6) if the system-wide invocation mechanism only accepts URIs.

The clipboard is another frequently used way to transfer URIs and IRIs from one application to another. On most platforms, the clipboard is able to store and transfer text in many languages and scripts. Correctly used, the clipboard transfers characters, not octets, which will do the right thing with IRIs.

8.4. URI/IRI Generation

Systems that offer resources through the Internet, where those resources have logical names, sometimes automatically generate URIs for the resources they offer. For example, some HTTP servers can generate a directory listing for a file directory and then respond to the generated URIs with the files.

Many legacy character encodings are in use in various file systems. Many currently deployed systems do not transform the local character representation of the underlying system before generating URIs.

For maximum interoperability, systems that generate resource identifiers should make the appropriate transformations. For example, if a file system contains a file named "résumé.html", a server should expose this as "r%C3%A9sum%C3%A9.html" in a URI, which allows use of "résumé.html" in an IRI, even if locally the file name is kept in a character encoding other than UTF-8.

This recommendation particularly applies to HTTP servers. For FTP servers, similar considerations apply; see [RFC2640].

8.5. URI/IRI Selection

In some cases, resource owners and publishers have control over the IRIs used to identify their resources. This control is mostly executed by controlling the resource names, such as file names, directly.

In these cases, it is recommended to avoid choosing IRIs that are easily confused. For example, for US-ASCII, the lower-case ell ("l") is easily confused with the digit one ("1"), and the upper-case oh ("O") is easily confused with the digit zero ("0"). Publishers should avoid confusing users with "br0ken" or "1ame" identifiers.

Outside the US-ASCII repertoire, there are many more opportunities for confusion; a complete set of guidelines is too lengthy to include here. As long as names are limited to characters from a single script, native writers of a given script or language will know best when ambiguities can appear, and how they can be avoided. What may look ambiguous to a stranger may be completely obvious to the average native user. On the other hand, in some cases, the UCS contains variants for compatibility reasons; for example, for typographic purposes. These should be avoided wherever possible. Although there may be exceptions, newly created resource names should generally be in NFKC [UTR15] (which means that they are also in NFC).

As an example, the UCS contains the "fi" ligature at U+FB01 for compatibility reasons. Wherever possible, IRIs should use the two letters "f" and "i" rather than the "fi" ligature. An example where the latter may be used is in the query part of an IRI for an explicit search for a word written containing the "fi" ligature.

In certain cases, there is a chance that characters from different scripts look the same. The best known example is the similarity of the Latin "A", the Greek "Alpha", and the Cyrillic "A". To avoid such cases, IRIs should only be created where all the characters in a single component are used together in a given language. This usually means that all of these characters will be from the same script, but there are languages that mix characters from different scripts (such as Japanese). This is similar to the heuristics used to distinguish between letters and numbers in the examples above. Also, for Latin, Greek, and Cyrillic, using lowercase letters results in fewer ambiguities than using uppercase letters would.

8.6. Display of URIs/IRIs

In situations where the rendering software is not expected to display non-ASCII parts of the IRI correctly using the available layout and font resources, these parts should be percent-encoded before being

displayed.

For display of Bidi IRIs, please see Section 4.1.

8.7. Interpretation of URIs and IRIs

Software that interprets IRIs as the names of local resources should accept IRIs in multiple forms and convert and match them with the appropriate local resource names.

First, multiple representations include both IRIs in the native character encoding of the protocol and also their URI counterparts.

Second, it may include URIs constructed based on character encodings other than UTF-8. These URIs may be produced by user agents that do not conform to this specification and that use legacy character encodings to convert non-ASCII characters to URIs. Whether this is necessary, and what character encodings to cover, depends on a number of factors, such as the legacy character encodings used locally and the distribution of various versions of user agents. For example, software for Japanese may accept URIs in Shift_JIS and/or EUC-JP in addition to UTF-8.

Third, it may include additional mappings to be more user-friendly and robust against transmission errors. These would be similar to how some servers currently treat URIs as case insensitive or perform additional matching to account for spelling errors. For characters beyond the US-ASCII repertoire, this may, for example, include ignoring the accents on received IRIs or resource names. Please note that such mappings, including case mappings, are language dependent.

It can be difficult to identify a resource unambiguously if too many mappings are taken into consideration. However, percent-encoded and not percent-encoded parts of IRIs can always be clearly distinguished. Also, the regularity of UTF-8 (see [Duerst97]) makes the potential for collisions lower than it may seem at first.

8.8. Upgrading Strategy

Where this recommendation places further constraints on software for which many instances are already deployed, it is important to introduce upgrades carefully and to be aware of the various interdependencies.

If IRIs cannot be interpreted correctly, they should not be created, generated, or transported. This suggests that upgrading URI interpreting software to accept IRIs should have highest priority.

On the other hand, a single IRI is interpreted only by a single or very few interpreters that are known in advance, although it may be entered and transported very widely.

Therefore, IRIs benefit most from a broad upgrade of software to be able to enter and transport IRIs. However, before an individual IRI is published, care should be taken to upgrade the corresponding interpreting software in order to cover the forms expected to be received by various versions of entry and transport software.

The upgrade of generating software to generate IRIs instead of using a local character encoding should happen only after the service is upgraded to accept IRIs. Similarly, IRIs should only be generated when the service accepts IRIs and the intervening infrastructure and protocol is known to transport them safely.

Software converting from URIs to IRIs for display should be upgraded only after upgraded entry software has been widely deployed to the population that will see the displayed result.

Where there is a free choice of character encodings, it is often possible to reduce the effort and dependencies for upgrading to IRIs by using UTF-8 rather than another encoding. For example, when a new file-based Web server is set up, using UTF-8 as the character encoding for file names will make the transition to IRIs easier. Likewise, when a new Web form is set up using UTF-8 as the character encoding of the form page, the returned query URIs will use UTF-8 as the character encoding (unless the user, for whatever reason, changes the character encoding) and will therefore be compatible with IRIs.

These recommendations, when taken together, will allow for the extension from URIs to IRIs in order to handle characters other than US-ASCII while minimizing interoperability problems. For considerations regarding the upgrade of URI scheme definitions, see Section 6.4.

9. IANA Considerations

RFC Editor and IANA note: Please Replace RFC XXXX with the number of this document when it issues as an RFC.

IANA maintains a registry of "URI schemes". A "URI scheme" also serves an "IRI scheme".

To clarify that the URI scheme registration process also applies to IRIs, change the description of the "URI schemes" registry header to say "[RFC4395] defines an IANA-maintained registry of URI Schemes.

These registries include the Permanent and Provisional URI Schemes. RFC XXXX updates this registry to designate that schemes may also indicate their usability as IRI schemes.

Update "per RFC 4395" to "per RFC 4395 and RFC XXXX".

10. Security Considerations

The security considerations discussed in [RFC3986] also apply to IRIs. In addition, the following issues require particular care for IRIs.

Incorrect encoding or decoding can lead to security problems. In particular, some UTF-8 decoders do not check against overlong byte sequences. As an example, a "/" is encoded with the byte 0x2F both in UTF-8 and in US-ASCII, but some UTF-8 decoders also wrongly interpret the sequence 0xC0 0xAF as a "/". A sequence such as "%C0%AF.." may pass some security tests and then be interpreted as "/.." in a path if UTF-8 decoders are fault-tolerant, if conversion and checking are not done in the right order, and/or if reserved characters and unreserved characters are not clearly distinguished.

There are various ways in which "spoofing" can occur with IRIs. "Spoofing" means that somebody may add a resource name that looks the same or similar to the user, but that points to a different resource. The added resource may pretend to be the real resource by looking very similar but may contain all kinds of changes that may be difficult to spot and that can cause all kinds of problems. Most spoofing possibilities for IRIs are extensions of those for URIs.

Spoofing can occur for various reasons. First, a user's normalization expectations or actual normalization when entering an IRI or transcoding an IRI from a legacy character encoding do not match the normalization used on the server side. Conceptually, this is no different from the problems surrounding the use of case-insensitive web servers. For example, a popular web page with a mixed-case name ("http://big.example.com/PopularPage.html") might be "spoofed" by someone who is able to create "http://big.example.com/popularpage.html". However, the use of unnormalized character sequences, and of additional mappings for user convenience, may increase the chance for spoofing. Protocols and servers that allow the creation of resources with names that are not normalized are particularly vulnerable to such attacks. This is an inherent security problem of the relevant protocol, server, or resource and is not specific to IRIs, but it is mentioned here for completeness.

Spoofing can occur in various IRI components, such as the domain name part or a path part. For considerations specific to the domain name part, see [RFC3491]. For the path part, administrators of sites that allow independent users to create resources in the same sub area may have to be careful to check for spoofing.

Spoofing can occur because in the UCS many characters look very similar. Details are discussed in Section 8.5. Again, this is very similar to spoofing possibilities on US-ASCII, e.g., using "br0ken" or "1ame" URIs.

Spoofing can occur when URIs with percent-encodings based on various character encodings are accepted to deal with older user agents. In some cases, particularly for Latin-based resource names, this is usually easy to detect because UTF-8-encoded names, when interpreted and viewed as legacy character encodings, produce mostly garbage.

When concurrently used character encodings have a similar structure but there are no characters that have exactly the same encoding, detection is more difficult.

Spoofing can occur with bidirectional IRIs, if the restrictions in Section 4.2 are not followed. The same visual representation may be interpreted as different logical representations, and vice versa. It is also very important that a correct Unicode bidirectional implementation be used.

The use of Legacy Extended IRIs introduces additional security issues.

11. Acknowledgements

For contributions to this update, we would like to thank Ian Hickson, Michael Sperberg-McQueen, Dan Connolly, Norman Walsh, Richard Tobin, Henry S. Thomson, and the XML Core Working Group of the W3C.

The discussion on the issue addressed here started a long time ago. There was a thread in the HTML working group in August 1995 (under the topic of "Globalizing URIs") and in the www-international mailing list in July 1996 (under the topic of "Internationalization and URLs"), and there were ad-hoc meetings at the Unicode conferences in September 1995 and September 1997.

For contributions to the previous version of this document, RFC 3987, many thanks go to Francois Yergeau, Matitiahu Allouche, Roy Fielding, Tim Berners-Lee, Mark Davis, M.T. Carrasco Benitez, James Clark, Tim Bray, Chris Wendt, Yaron Goland, Andrea Vine, Misha Wolf, Leslie

Daigle, Ted Hardie, Bill Fenner, Margaret Wasserman, Russ Housley, Makoto MURATA, Steven Atkin, Ryan Stansifer, Tex Texin, Graham Klyne, Bjoern Hoehrmann, Chris Lilley, Ian Jacobs, Adam Costello, Dan Oscarson, Elliotte Rusty Harold, Mike J. Brown, Roy Badami, Jonathan Rosenne, Asmus Freytag, Simon Josefsson, Carlos Viegas Damasio, Chris Haynes, Walter Underwood, and many others.

A definition of HyperText Reference was initially produced by Ian Hixson, and further edited by Dan Connolly and C. M. Spergerg-McQueen.

Thanks to the Internationalization Working Group (I18N WG) of the World Wide Web Consortium (W3C), and the members of the W3C I18N Working Group and Interest Group for their contributions and their work on [CharMod]. Thanks also go to the members of many other W3C Working Groups for adopting IRIs, and to the members of the Montreal IAB Workshop on Internationalization and Localization for their review.

12. Open Issues

NOTE: The issues noted in this section should be addressed before the document is submitted as an RFC. These issues are not in any particular order, and do not necessarily form a complete list of all known issues.

length limits on domain name See, for example, http://lists.w3.org/Archives/Public/public-iri/2009Sep/0064.html discussion on public-iri@w3.org (that discussion is mostly irrelevant now as the "63 octets in UTF-8 per label" restriction was dropped)

Allow generic scheme-independent IRI to URI translation Previous drafts of this specification proposed a generic IRI to URI transformation using pct-encoding, and allowed domain name translation to be optionally handled by retranslating host names from pct-encoding back into Unicode and then into punycode. This draft does not allow that behavior, but this should be fixed to be in line with RFC 3986 syntax and to lead implementations towards an uniform an long-term URI<->IRI correspondence. See also [Gettys]

update URI scheme registry? This document starts the process of making minor changes to the URI scheme registry. This should be handled as an update to RFC 4395.

utf8 in HTTP Not really IRI issue, but some HTTP implementations send UTF8 path directly, review.

handling of \\ Some web applications convert \ to / and others don't. Make this mandatory or disallowed (but not optional), for Web Addresses.

dealing with disallowed IRI characters

misplaced text Find a place to note that some older software transcoding to UTF-8 may produce illegal output for some input, in particular for characters outside the BMP (Basic Multilingual Plane). As an example, for the IRI with non-BMP characters (in XML Notation):
"http://example.com/𐌀𐌁&#x10302";
which contains the first three letters of the Old Italic alphabet, the correct conversion to a URI is
"http://example.com/%F0%90%8C%80%F0%90%8C%81%F0%90%8C%82"

Special Query Handling needed? The percent-encoding handling of query components in the HTTP scheme is really unfortunate. There is no good normative advice to give if the percent-encoding is delayed until the query-IRI is interpreted. Could HTML ask browsers to percent-encode the form data using the document character set BEFORE the query IRI is constructed, and only in the case where the document character set isn't Unicode-based and the query is being added to http: or https: URIs? This would give more consistent results. Browsers might have to change their behavior in constructing the IRI-with-query-added, but the results would be more consistent and fewer bugs, and it wouldn't affect interpretation of any existing web pages. It would remove the need to have a normative special case for queries in HTML documents, just for http, in a way in which things like transcoding etc. wouldn't work well. You could tell the difference between a query URI in the address bar and one created via a form because the address bar would always be UTF-8. The browsers might have to change the algorithm for showing the address in the adress bar to know how to undo the encoding.

handling illegal characters Section 3.3 used to apply only to characters in either 'ucschar' or 'iprivate', but then later said that systems accepting IRIs MAY also deal with the printable characters in US-ASCII that are not allowed in URIs, namely "<", ">", '"', space, "{", "}", "|", "\", "^", and "`". Larry felt that this a MAY would result in non-uniform behavior, because some systems would produce valid URI components and others wouldn't. Non-printable US-ASCII characters should be stripped by most software, so if they get to if they're passed on somewhere as IRI

characters, encoding them makes sense. The section also used to say "If these characters are found but are not converted, then the conversion SHOULD fail." but there is no notion of conversion failing -- every string is converted. Please note that the number sign ("#"), the percent sign ("%"), and the square bracket characters ("[", "]") are not part of the above list and MUST NOT be converted.

adding single % and hash Changed the BNF to not match the URI document in allowing single % in path but not everywhere, and allowing a # in the fragment part.

13. Change Log

Note to RFC Editor: Please completely remove this section before publication.

13.1. Changes from draft-duerst-iri-bis-07 to draft-ietf-iri-3987bis-00

Changed draft name, date, last paragraph of abstract, and titles in change log, and added this section in moving from draft-duerst-iri-bis-07 (personal submission) to draft-ietf-iri-3987bis-00 (WG document).

13.2. Changes from -06 to -07 of draft-duerst-iri-bis

Major restructuring of IRI processing model to make scheme-specific translation necessary to handle IDNA requirements and for consistency with web implementations.

Starting with IRI, you want one of:

a IRI components (IRI parsed into UTF8 pieces)

b URI components (URI parsed into ASCII pieces, encoded correctly)

c whole URI (for passing on to some other system that wants whole URIs)

13.2.1. OLD WAY

1. Pct-encoding on the whole thing to a URI. (c1) If you want a (maybe broken) whole URI, you might stop here.

2. Parsing the URI into URI components. (b1) If you want (maybe broken) URI components, stop here.

3. Decode the components (undoing the pct-encoding). (a) if you want IRI components, stop here.

4. reencode: Either using a different encoding some components (for domain names, and query components in web pages, which depends on the component, scheme and context), and otherwise using pct-encoding. (b2) if you want (good) URI components, stop here.

5. reassemble the reencoded components. (c2) if you want a (*good*) whole URI stop here.

13.2.2. NEW WAY

1. Parse the IRI into IRI components using the generic syntax. (a) if you want IRI components, stop here.

2. Encode each components, using pct-encoding, IDN encoding, or special query part encoding depending on the component scheme or context. (b) If you want URI components, stop here.

3. reassemble the a whole URI from URI components. (c) if you want a whole URI stop here.

13.3. Changes from -05 to -06 of draft-duerst-iri-bis

- Add HyperText Reference, change abstract, acks and references for it
- Add Masinter back as another editor.
- Masinter integrates HRef material from HTML5 spec.
- Rewrite introduction sections to modernize.

13.4. Changes from -04 to -05 of draft-duerst-iri-bis

- Updated references.
- Changed IPR text to pre5378Trust200902.

13.5. Changes from -03 to -04 of draft-duerst-iri-bis

- Added explicit abbreviation for LEIRIs.
- Mentioned LEIRI references.
- Completed text in LEIRI section about tag characters and about specials.

13.6. Changes from -02 to -03 of draft-duerst-iri-bis

- Updated some references.
- Updated Michel Suginard's coordinates.

13.7. Changes from -01 to -02 of draft-duerst-iri-bis

- Added tag range to iprivate (issue private-include-tags-115).
- Added Specials (U+FFF0-FFFD) to Legacy Extended IRIs.

13.8. Changes from -00 to -01 of draft-duerst-iri-bis

- Changed from "IRIs with Spaces/Controls" to "Legacy Extended IRI" based on input from the W3C XML Core WG. Moved the relevant subsections to the back and promoted them to a section.
- Added some text re. Legacy Extended IRIs to the security section.
- Added a IANA Consideration Section.
- Added this Change Log Section.
- Added a section about "IRIs with Spaces/Controls" (converting from a Note in RFC 3987).

13.9. Changes from RFC 3987 to -00 of draft-duerst-iri-bis

Fixed errata (see http://www.rfc-editor.org/cgi-bin/erratasearch.pl/doc/html/rfc3987).

14. References

14.1. Normative References

[ASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[ISO10646] International Organization for Standardization, "ISO/IEC 10646:2003: Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646, December 2003.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[STD68] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[UNI9] Davis, M., "The Bidirectional Algorithm", Unicode Standard Annex #9, March 2004, <http://www.unicode.org/reports/tr9/tr9-13.html>.

[UNIV4] The Unicode Consortium, "The Unicode Standard, Version 5.1.0, defined by: The Unicode Standard, Version 5.0 (Boston, MA, Addison-Wesley, 2007. ISBN 0-321-48091-0), as amended by Unicode 4.1.0 (http://www.unicode.org/versions/Unicode5.1.0/)", April 2008.

[UTR15] Davis, M. and M. Duerst, "Unicode Normalization Forms", Unicode Standard Annex #15, March 2008, <http://www.unicode.org/unicode/reports/tr15/tr15-23.html>.

14.2. Informative References

[BidiEx] "Examples of bidirectional IRIs", <http://www.w3.org/International/iri-edit/BidiExamples>.

[CharMod] Duerst, M., Yergeau, F., Ishida, R., Wolf, M., and T. Texin, "Character Model for the World Wide Web: Resource Identifiers", World Wide Web Consortium Candidate Recommendation, November 2004, <http://www.w3.org/TR/charmod-resid>.

[Duerst97]

Duerst, M., "The Properties and Promises of UTF-8", Proc. 11th International Unicode Conference, San Jose , September 1997, <http://www.ifi.unizh.ch/mml/mduerst/papers/PDF/IUC11-UTF-8.pdf>.

[Gettys] Gettys, J., "URI Model Consequences", <http://www.w3.org/DesignIssues/ModelConsequences>.

[HTML4] Raggett, D., Le Hors, A., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation, December 1999, <http://www.w3.org/TR/html401/appendix/notes.html#h-B.2>.

[HTML5] Hickson, I. and D. Hyatt, "A vocabulary and associated APIs for HTML and XHTML", World Wide Web Consortium Working Draft, April 2009, <http://www.w3.org/TR/2009/WD-html5-20090423/>.

[LEIRI] Thompson, H., Tobin, R., and N. Walsh, "Legacy extended IRIs for XML resource identification", World Wide Web Consortium Note, November 2008, <http://www.w3.org/TR/leiri/>.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2130] Weider, C., Preston, C., Simonsen, K., Alvestrand, H., Atkinson, R., Crispin, M., and P. Svanberg, "The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996", RFC 2130, April 1997.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC2192] Newman, C., "IMAP URL Scheme", RFC 2192, September 1997.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC2368] Hoffman, P., Masinter, L., and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[RFC2384] Gellens, R., "POP URL Scheme", RFC 2384, August 1998.

[RFC2396] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform

Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2640] Curtin, B., "Internationalization of the File Transfer Protocol", RFC 2640, July 1999.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35, RFC 4395, February 2006.

[UNIXML] Duerst, M. and A. Freytag, "Unicode in XML and other Markup Languages", Unicode Technical Report #20, World Wide Web Consortium Note, June 2003, <http://www.w3.org/TR/unicode-xml/>.

[XLink] DeRose, S., Maler, E., and D. Orchard, "XML Linking Language (XLink) Version 1.0", World Wide Web Consortium Recommendation, June 2001, <http://www.w3.org/TR/xlink/#link-locators>.

[XML1] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Forth Edition)", World Wide Web Consortium Recommendation, August 2006, <http://www.w3.org/TR/REC-xml>.

[XMLNamespace]
Bray, T., Hollander, D., Layman, A., and R. Tobin, "Namespaces in XML (Second Edition)", World Wide Web Consortium Recommendation, August 2006, <http://www.w3.org/TR/REC-xml-names>.

[XMLSchema]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", World Wide Web Consortium Recommendation, May 2001, <http://www.w3.org/TR/xmlschema-2/#anyURI>.

[XPointer]
Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer Framework", World Wide Web Consortium Recommendation, March 2003, <http://www.w3.org/TR/xptr-framework/#escaping>.

Appendix A. Design Alternatives

This section briefly summarizes some design alternatives considered earlier and the reasons why they were not chosen.

A.1. New Scheme(s)

Introducing new schemes (for example, httpi:, ftpi:,...) or a new metascheme (e.g., i:, leading to URI/IRI prefixes such as i:http:, i:ftp:,...) was proposed to make IRI-to-URI conversion scheme dependent or to distinguish between percent-encodings resulting from IRI-to-URI conversion and percent-encodings from legacy character encodings.

New schemes are not needed to distinguish URIs from true IRIs (i.e., IRIs that contain non-ASCII characters). The benefit of being able to detect the origin of percent-encodings is marginal, as UTF-8 can be detected with very high reliability. Deploying new schemes is extremely hard, so not requiring new schemes for IRIs makes deployment of IRIs vastly easier. Making conversion scheme dependent is highly inadvisable and would be encouraged by separate schemes for IRIs. Using a uniform convention for conversion from IRIs to URIs makes IRI implementation orthogonal to the introduction of actual new schemes.

A.2. Character Encodings Other Than UTF-8

At an early stage, UTF-7 was considered as an alternative to UTF-8 when IRIs are converted to URIs. UTF-7 would not have needed percent-encoding and in most cases would have been shorter than percent-encoded UTF-8.

Using UTF-8 avoids a double layering and overloading of the use of the "+" character. UTF-8 is fully compatible with US-ASCII and has therefore been recommended by the IETF, and is being used widely.

UTF-7 has never been used much and is now clearly being discouraged. Requiring implementations to convert from UTF-8 to UTF-7 and back would be an additional implementation burden.

A.3. New Encoding Convention

Instead of using the existing percent-encoding convention of URIs, which is based on octets, the idea was to create a new encoding convention; for example, to use "%u" to introduce UCS code points.

Using the existing octet-based percent-encoding mechanism does not need an upgrade of the URI syntax and does not need corresponding

server upgrades.

A.4. Indicating Character Encodings in the URI/IRI

Some proposals suggested indicating the character encodings used in an URI or IRI with some new syntactic convention in the URI itself, similar to the "charset" parameter for e-mails and Web pages. As an example, the label in square brackets in "http://www.example.org/ros[iso-8859-1]é" indicated that the following "é" had to be interpreted as iso-8859-1.

If UTF-8 is used exclusively, an upgrade to the URI syntax is not needed. It avoids potentially multiple labels that have to be copied correctly in all cases, even on the side of a bus or on a napkin, leading to usability problems (and being prohibitively annoying). Exclusively using UTF-8 also reduces transcoding errors and confusion.

Authors' Addresses

Martin Duerst (Note: Please write "Duerst" with u-umlaut wherever possible, for example as "Dürst" in XML and HTML.)
Aoyama Gakuin University
5-10-1 Fuchinobe
Sagamihara, Kanagawa 229-8558
Japan

Phone: +81 42 759 6329
Fax: +81 42 759 6495
Email: mailto:duerst@it.aoyama.ac.jp
URI: http://www.sw.it.aoyama.ac.jp/D%C3%BCrst/
(Note: This is the percent-encoded form of an IRI.)

Michel Suignard
Unicode Consortium
P.O. Box 391476
Mountain View, CA 94039-1476
U.S.A.

Phone: +1-650-693-3921
Email: mailto:michel@unicode.org
URI: http://www.suignard.com

Larry Masinter
Adobe
345 Park Ave
San Jose, CA 95110
U.S.A.

Phone: +1-408-536-3024
Email: mailto:masinter@adobe.com
URI: http://larry.masinter.net