

ICNRG
Internet-Draft
Intended status: Experimental
Expires: October 8, 2016

M. Mosko
C. Wood
PARC, Inc.
April 6, 2016

The CCNx URI Scheme
draft-mosko-icnrg-ccnxurischeme-01

Abstract

This document defines an [RFC3986](#) URI compliant identifier called a Labeled Segment URI in which name segments carry a label. This allows differentiation between unrelated resources with similar identifiers. This document also specifies the CCNx URI scheme, called "ccnx:," which conforms to the labeled segment encoding rules presented here. The CCNx URI scheme applies specific labels to each name segment of a URI to disambiguate between resources with similar names. This document defines a specific set of segment labels with label semantics.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

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

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

This Internet-Draft will expire on October 8, 2016.

Copyright Notice

Copyright (c) 2016 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

Internet-Draft

CCNxScheme

April 2016

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Requirements Language	4
2.	URI path segment grammar for label=value pairs	5
2.1.	Labeled Segments	5
2.2.	URI comparison	7
3.	Application to CCNx Names	9
3.1.	The ccnx Scheme	9
3.2.	URI Representation	9
3.2.1.	Examples	11
3.3.	ccnx: URI comparison	11
4.	IRI Considerations	13
5.	Acknowledgements	14
6.	IANA Considerations	15
7.	Security Considerations	16
8.	References	17
8.1.	Normative References	17
8.2.	Informative References	17
	Authors' Addresses	18

1. Introduction

A Labeled Segment is an URI [[RFC3986](#)] compliant convention that allows an application or protocol to embed labels in name segments, thus disambiguating the resource identified by the path. Labeled Segment URIs also allow for query and fragment components to follow the Labeled Segment form.

Some protocols may wish to disambiguate name segments between different identifier spaces, such as "version" and "page". Other protocols may wish to use a type system such as "/str=parc/int=7" and "/str=parc/str=7". Labeled Segment URIs provide an unambiguous and flexible representation in systems that allow resources with otherwise similar names.

It is not sufficient to leave the determination of type to application-specific conventions. In a networked system with multiple applications accessing resources generated by other applications, there needs to be a set of common conventions. For example, if one application uses a base 64 encoding of a frame number, e.g. base64(0xbdea), and another uses "ver=" to represent a document version, there is an ambiguity because base64(0xbdea) is the string "ver=".

Labeled Segments defines "ls-segment" as "label[:param]=value", where the value only contains unreserved, percent-encoded, or certain sub-delim characters. In the previous example, one application would say "/frame=%BD%EA" and the other would say "/ver=".

In this document, we use URI [[RFC3986](#)] terminology, therefore a URI and CCNx Name are both composed of a URI path, which is a collection of name segments. We do not use the term "name component" as was common in old CCNx. In this document, the word "segment" alone means "name segment."

URIs conforming to the CCNx URI scheme carry a label for each name

segment. The contents of each name segment must conform to the label semantics. Example segment types are "Binary Segment", "Name", and "KeyId".

We use Labeled Segment URIs as the canonical, human-readable representation. There is an unambiguous, one-to-one correspondence between an absolute LS-URI path and a Labeled Name. Relative URI representations are removed during encoding, so no relative name ends up in wire format. Some labels are URIs that are IRI [[RFC3987](#)] compatible.

Labeled Names shall be used everywhere a Name is used in CCNx, such

as in the Name of an Interest or Content Object. They are also used in Links, KeyLocators, or any other place requiring a name. When encoded for the wire, a binary representation is used, depending on the specific wire format codec, which is outside the scope of this document.

This document specifies:

- o the ccnx scheme.
- o a canonical URI representation.

Formal grammars use the ABNF [[RFC5234](#)] notation.

TODD: We have not adopted Requirements Language yet.

[1.1](#). Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

[2.](#) URI path segment grammar for label=value pairs

[2.1.](#) Labeled Segments

This section describes the formal grammar for Labeled Segments using ABNF [[RFC5234](#)] notation. We do not impose restrictions on the length of labels or values. The semantics of values are URI scheme specific, here we only describe the meta-structure of Labeled Segments. We begin by reviewing some definitions from [[RFC3986](#)] that define an absolute path URI.

```
URI           = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part     = "://" authority path-abempty
               / path-absolute
               / <other path types>
path-absolute = "/" [ segment-nz *( "/" segment ) ]
segment       = *pchar
segment-nz    = 1*pchar
pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"
query         = *( pchar / "/" / "?" )
fragment      = *( pchar / "/" / "?" )
pct-encoded   = "%" HEXDIG HEXDIG
```

```

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

```

Labeled Segments defines a new segment type that provides unambiguous representation of a segment's label and its value. We define the top-level LS-URI as the same form as a URI, wherein each part conforms to the Label Segment grammar, which is a subset of the URI grammar.

```

LS-URI        = scheme ":" ls-hier-part ["?" ls-query]
               ["#" fragment]
ls-hier-part   = ["//" authority] ls-path-absolute
ls-path-absolute = "/" [ first-segment *( "/" ls-segment ) ]
first-segment  = ls-segment-nz
ls-segment     = lpv-segment / v-segment
lpv-segment    = label [":" param] "=" *s-value-nz
v-segment      = *s-value-nz
ls-segment-nz  = lpv-segment-nz / v-segment-nz
lpv-segment-nz = label [":" param] "=" s-value-nz
v-segment-nz   = s-value-nz
label          = alpha-t / num-t
param          = alpha-t / num-t
s-value-nz     = 1*(s-pchar)

```

```

ls-query          = *1 ( lpv-component / v-component
                        *( "&" (lpv-component / v-component) ) )
lpv-component     = label [":" param] "=" q-value
v-component       = q-value
q-value           = *(q-pchar)

alpha-t           = ALPHA *(ALPHA / DIGIT)
num-t             = dec-t / hex-t
dec-t             = 1*(DIGIT)
hex-t             = "0x" 1*(HEXDIG)
ls-pchar          = unreserved / pct-encoded / ls-sub-delims
s-pchar           = ls-pchar / ":" / "@" / "&"
q-pchar           = ls-pchar / ":" / "@" / "/"
ls-sub-delims     = "!" / "$" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";"

```

A Labeled Segment URI (LS-URI) contains a scheme that uses Labeled Segments, an optional authority, a labeled segment absolute path (ls-path-aboslute), an optional labeled segment query (ls-query), and a fragment. The authority is URI scheme specific and the fragment is independent of the URI scheme.

the ls-path-aboslute is a first-segment followed by zero or more "/" ls-segment. The first-segment may be empty or a non-zero ls-segment (ls-segment-nz). If it is empty, it corresponds to a 0-lenght name which typically is a default route. It is distinct from a 1-segment name with no value (which is not allowed). The first-segment MUST either be empty (the 0-lenght name) or MUST have a value. If the first-segment is an ls-segment-nz, then it will have a value.

An ls-segment may (lpv-segment) or may not (v-segment) have a label. A particular LS-URI scheme MUST define how unlabeled segments are processed, and MAY disallow them. A v-segment is an implied type.

Once the implied type is resolved, it functions like an lpv-segment.

An lpv-segment has a label, optional parameter, and optional value (s-value-nz). An empty value is a 0-length name segment with a defined type. This is distinct from a 0-length first-segment, which has neither type nor value.

lpv-segment values come from the s-pchar set, which excludes the "="

equal sign. This means that the only equal sign in a name segment must be the delimiter between the label:param and the value. Within the value, an equal sign must be percent encoded.

lpv-segment labels and values may be alpha-numeric identifiers or numbers (decimal or hexadecimal). For example, one scheme may define the labels "name", "version", and "frame". A version may be of types "date" or "serial", meaning that the version is either a date or a monotonic serial number. Some examples of resulting LS-URIs are: "/name=parc/name=csl/version:date=20130930" or "/name=alice_smith/version:serial=299". The parameters may also indicate an instance of a label, such as "/name=books/year:1=1920/year:3=1940", where there are scheme or application semantics associated with "year:1" and "year:3".

lpv-segment labels and parameters may also be numbers. For example, a protocol with a binary and URI representation may not have pre-defined all possible labels. In such cases, it could render unknown labels as their binary value, such as "/name=marc/x2003=green".

The ls-query component is a non-hierarchical set of components separated by "&". Each ls-query component is either a lpv-component or a v-component, similar to segments. They are based on q-value, which uses q-pchar that excludes "&", but includes "/". This allows an LS-URI scheme to use type query parameters.

Labeled Segments allow for dot-segments "." and ".." in a v-segment. They operate as normal. A single dot "." refers to the current hierarchy level and may be elided when the URI is resolved. Double dot ".." segments pop off the previous non-dot segment. An lpv-segment with a value of "." or ".." is not a dot-segment. It means that the value of the given label is "." or "..". For example /a=parc/b=csl/.. is equivalent to "/a=parc/b=csl", but the LS-URI "/a=parc/b=csl/c=.." does not contain a dot-segment.

[2.2](#). URI comparison

An LS-URI scheme MUST specify the normalization rules to be used, following the methods of [Section 6 \[RFC3986\]](#). At minimum, an LS-URI scheme SHOULD do the following:

- o Normalize unrestricted percent-encodings to the unrestricted form.

- o Normalize num-t to either dec-t or hex-t.
- o If the scheme allows for value-only segments or query components and interprets them as a default type, they should be normalized to having the type specified.
- o If the scheme allows for undefined labels and represents them, for example, as num-t, then it should normalize all labels to their corresponding num-t. If "name", for example, is known to be %x50 in a binary encoding of the URI, then all labels should be compared using their numeric value.

[3.](#) Application to CCNx Names

[3.1.](#) The ccnx Scheme

This section describes the CCNx URI scheme "ccnx:" for Labeled Names. A Labeled Name assigns a semantic type or label to each segment of the hierarchical content Name.

Unless otherwise specified, a name segment is an arbitrary sequence of octets.

Several name segment labels are binary unsigned integers. These are always encoded as variable length sequences of 1 or more octets in network byte order using the shortest representation (i.e. no leading %x00). The value of "0" is encoded as the single byte of "%x00". A zero-length sequence must be interpreted as "not present."

The CCNx Name segment types are:

- o Name Segment: A generic name segment that includes arbitrary octets.
- o Application Type N: An application may use application-specific parameters, numbered as integers, where N is from 0 to a system-specific maximum, not less than 255. These are represented as "App:1=value", for example.

It is common for an information centric networking protocol, such as CCNx or NDN, to use a binary on-the-wire representation for messages. Such protocols, if they use the ccnx: scheme, must have an appropriate codec that unambiguously represents Labeled Content Information in the chosen wire format. Relative dot-segments should not occur in the wire format, they should be resolved before encoding.

[3.2.](#) URI Representation

Typed Names use a standard [RFC 3986](#) representation following the LS-URI convention. A name segment consists of any "unreserved" characters plus percent-encoded characters. Reserved characters must be percent encoded.

Within an absolute path, each segment consists of an "ls-segment" (c.f. LS-URI). A labeled segment is a type and a name component value, with a URI representation of "type=value". The "type=" portion may be omitted if it is type Name.

Some name types take a parameter, such as the Application types.

They are represented as "A:nnn=value", where the "nnn" is the application type number and value is the name component.

A CCNx URI MUST NOT include an Authority, Query, or Fragment. It is an error to include them.

Dot-segments (relative name components) are resolved when the URI is converted to a Typed Name. The "." dot-segment is removed. The ".." dot-segment is removed along with the previous non-dot-segment.

Type	Display	Name
'Name'	Hexadecimal	Name Segment
'IPID'	Hexadecimal	Interest Payload Identifier segment
'App:0' - 'App:255'	Hexadecimal	Application Component

Table 1: The CCNx URI Scheme Types

[3.2.1.](#) Examples

A name / is
ccnx:/ and is a 0-length name.

A name /Name= is
ccnx:/Name= and is a 1-segment name of 0-length.

A name /foo/bar.
ccnx:/Name=foo/Name=bar
ccnx:/foo/Name=bar
ccnx:/foo/bar

A name /foo/bar with key %xA0.
ccnx:/Name=foo/Name=bar/App:1=0xA0

A name /foo/bar with version %xA0 and App:2 value 0x09.
ccnx:/foo/bar/Version=0xA0/App:2=0x09

A name /foo/.., where the ".." is a literal name component,
not a relative dot-segment.
ccnx:/foo/Name=..

A name /foo/bar with application type 0 "hello"
and application type 1 "world".
ccnx:/Name=foo/Name=bar/App:0=hello/App:1=world

[3.3.](#) ccnx: URI comparison

While most comparisons are done using a wire format representation of a ccnx: URI, some applications may compare the CCNx URI using their URI representation. This section defines the rules for comparing

ccnx: URIs using the methods of [Section 6 \[RFC3986\]](#)

Comparing typed name URIs must be done with:

- o Syntax-based normalization
- o Case normalization: normalize the representation of percent encodings. ccnx: does not use the host portion of the URI, and should be ignored if present.
- o Percent encoding normalization: Percent encodings of unreserved characters must be converted to the unreserved character.
- o Path segment normalization: dot-segments must be resolved first.

- o Scheme-based normalization: The authority should be removed and the path represented as an absolute path.
- o Protocol-based normalization: Should not be done. A trailing slash indicates a zero-length terminal name component and signifies a different name.
- o typed-name-segment normalization: All segments should be presented with their type, do not elide the "N=" for Name components.
- o Binary unsigned integer normalization: remove any leading %x00 from numbers, leaving only the terminal %x00 for "0".
- o type parameters: they must have their percent encodings normalized. If they are integers, such as for the 'A' type, they must not have leading zeros.

[4.](#) IRI Considerations

International Resource Identifiers extend the unreserved character set to include characters above U+07F and encode them using percent encoding. This extension is compatible with the ccnx: schema. It applies only to the "value" portion of an ls-segment.

The canonical name is determined by the URI representation of the IRI, after applying the rules of [Section 3.1 of \[RFC3987\]](#) and resolving dot-segments. The canonical name thus includes the URI representation of language markers, including the bidirectional components.

The value of a UTF-8 Name segment should be interpreted using IRI rules, including bidirectional markers. They may be displayed using localized formats.

Binary unsigned integer types are not interpreted under IRI rules,

they are specifically percent encoded numbers. They may be displayed using a localized format.

[5.](#) Acknowledgements

[6.](#) IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs [[RFC5226](#)] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

7. Security Considerations

All drafts are required to have a security considerations section.
See [RFC 3552](#) [[RFC3552](#)] for a guide.

Internet-Draft

CCNxScheme

April 2016

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/[RFC2119](#), March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

8.2. Informative References

- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", [BCP 72](#), [RFC 3552](#), DOI 10.17487/RFC3552, July 2003, <<http://www.rfc-editor.org/info/rfc3552>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, [RFC 3986](#), DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", [RFC 3987](#), DOI 10.17487/RFC3987, January 2005, <<http://www.rfc-editor.org/info/rfc3987>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, [RFC 5234](#), DOI 10.17487/[RFC5234](#), January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

Internet-Draft

CCNxScheme

April 2016

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Christopher A. Wood
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4421
Email: christopher.wood@parc.com

