

Labeled Segments and the CCNx URI Scheme

April 2016 @ IETF 95
[draft-mosko-icnrg-ccnxurischeme-01](https://datatracker.ietf.org/doc/draft-mosko-icnrg-ccnxurischeme-01)

Overview

- In CCNx names, each path segment may have a distinct type.
- Specifies a way to represent this in a URI.
- Submitted Jan 2015, updated June 2015.
- Latest update April 2016 changed schema to ccnx: from lci: and fixed an error in the grammar.

In a nutshell

- General URI derived syntax
 - Reserve the “=” character.
 - Must be percent encoded if appears.
 - Specify that each path segment is of the form label=value.
 - Does not specify any labels.
- ccnx: scheme
 - Specifies labels
 - Specifies normalization rules

Labeled Segment (LS) URIs

- Similar to URIs defined RFC3986
- A LS-URI has a scheme, hierarchical part, optional query, and optional fragment

```
LS-URI = scheme ":" ls-hier-part ["?" ls-query] ["#" fragment]
```

LS-URI Hierarchical Parts

- Composed of an absolute path
- The first segment, if present, is special: it must be a labeled segment **with a non-zero length**.

```
ls-segment-nz  = lpv-segment-nz / v-segment-nz  
lpv-segment-nz = label [ ":" param ] "=" s-value-nz
```

- The remaining segments can be labeled segments or plain values and may have zero length

```
ls-segment = lpv-segment / v-segment
```

Implicit labels

- The LS-URI scheme allows the omission of the label and “=” so one may write simple URIs.
- In this case, a specific scheme **MUST** specify what the implicit label is.
- URIs with implicit labels have labels and when displayed are displayed with the label
 - E.g. ccnx:/foo/bar → ccnx:/NAME=foo/NAME=bar

The first path segment

- `ls-uri:/` (OK)
 - This is a name with no segments. It is not an implicit label.
- `ls-uri:/label=` (NOT ALLOWED)
 - This is a name with 1 segment and no value. This could be an implicit label with no value (case 1) so to avoid ambiguity is not allowed.
- `ls-uri:/label=foo/` (OK)
 - 2 segment name with second segment implicit and empty → `ls-uri:/label=foo/implicit=`

LS Queries and Fragments

- A query is composed of at least one labeled or value segment
- More segments may be appended with the proper ‘&’ concatenation marker

```
ls-query = *1 ( lpv-component / v-component  
                *( "&" (lpv-component / v-component) ) )
```

- Fragments are as defined in RFC3986

Normalizing URIs

- LS URIs must be normalized (using the methods in RFC3986 before comparison) such that:
 - Unrestricted percent-encodings are restricted
 - Numerical values are decimal or hexadecimal
 - Value-only segments or components that are assigned a default type should be normalized to a that type
 - Unknown labels should be normalized to a default value (per the scheme)

The CCNx URI Scheme

Overview

- The CCNx URI scheme is denoted “ccnx:”
 - The previous Labeled Content Identifier “lci:” scheme is deprecated
- The scheme specifies types for labeled segments
 - Name segment: a generic name that includes arbitrary octets
 - Payload identifier: a unique identifier for a CCNx message (interest)
 - Application type N: an application-specific type

Restrictions

- CCNx URIs MUST NOT include an Authority, Query, or Fragment
 - Names with these components are treated as errors

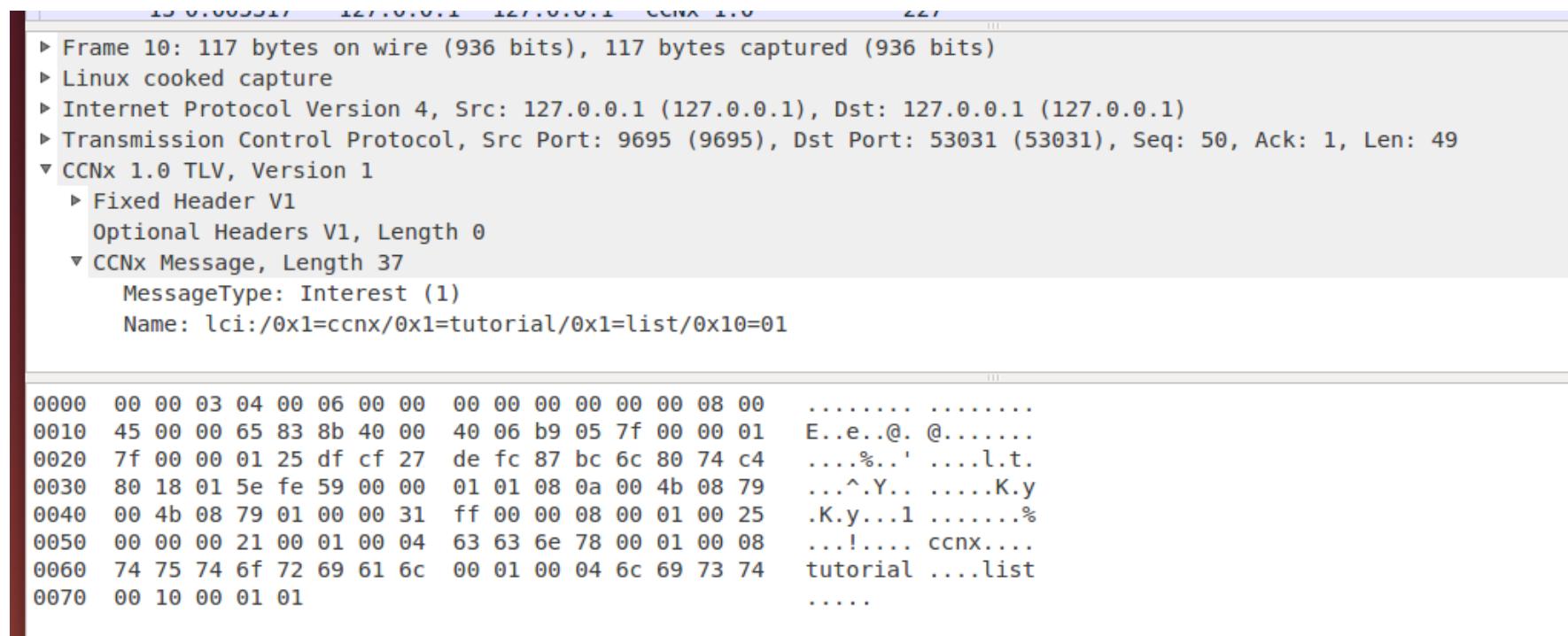
More on Labels

- Name labels are denoted as “Name=value”
 - This is the implicit type for unlabeled segments
- Payload ID labels are denoted as “IPID=value”
- Application-specific labels are denoted as “App:N=value”, where N = 0,...,255

Examples

Simple Representation	Canonical Representation	Description
/	ccnx:/	0-length name
/Name=	ccnx:/Name	1-segment name of 0-length
/foo/bar	ccnx:/Name=foo/Name=bar	Equivalent to ccnx:/foo/Name=bar ccnx:/foo/bar Etc.
/foo/bar with App:2=%x09	ccnx:/Name=foo/Name=bar/App:2=0x09	
/foo/bar with App:1 %xA0 and App:2 value 0x09	ccnx:/Name=foo/Name=bar/App:1=0xA0/App:2=0x09	

Wireshark Example



The screenshot shows a Wireshark capture window. The packet details pane displays a single frame (Frame 10) containing a CCNx 1.0 TLV message. The message is identified as an Interest (1) with the name "lci:/0x1=ccnx/0x1=tutorial/0x1=list/0x10=01". The bytes pane shows the raw hex and ASCII data of the message, which includes the CCNx header and the interest payload.

```
► Frame 10: 117 bytes on wire (936 bits), 117 bytes captured (936 bits)
► Linux cooked capture
► Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
► Transmission Control Protocol, Src Port: 9695 (9695), Dst Port: 53031 (53031), Seq: 50, Ack: 1, Len: 49
▼ CCNx 1.0 TLV, Version 1
  ► Fixed Header V1
    Optional Headers V1, Length 0
  ▼ CCNx Message, Length 37
    MessageType: Interest (1)
    Name: lci:/0x1=ccnx/0x1=tutorial/0x1=list/0x10=01

0000  00 00 03 04 00 06 00 00  00 00 00 00 00 00 08 00  .....
0010  45 00 00 65 83 8b 40 00  40 06 b9 05 7f 00 00 01  E..e..@. @.....
0020  7f 00 00 01 25 df cf 27  de fc 87 bc 6c 80 74 c4  ....%..'^....l.t.
0030  80 18 01 5e fe 59 00 00  01 01 08 0a 00 4b 08 79  ...^Y.. .....K.y
0040  00 4b 08 79 01 00 00 31  ff 00 00 08 00 01 00 25  .K.y...1 ....%
0050  00 00 00 21 00 01 00 04  63 63 6e 78 00 01 00 08  ...!.... ccnx....
0060  74 75 74 6f 72 69 61 6c  00 01 00 04 6c 69 73 74  tutorial ....list
0070  00 10 00 01 01          .....
```

Comparing CCNx Names

- Two options: compare binary-encoded versions or normalize and compare URI string versions
- Normalization rules include (at minimum):
 - Case normalization
 - Percent encoding normalization
 - Percent encodings of unreserved characters must be converted to the unreserved character
 - Path segment normalization
 - Must be resolved first
 - ...

Construction from Code

- Default name

```
CCNxName *name1 = ccnxName_CreateFromCString("ccnx:/");  
CCNxName *name2 = ccnxName_CreateFromCString("lci:/");
```

Output: "ccnx:/"

- With %-encoding

```
char *str = ccnx:/test/Name=MiISAg%3D%3D";  
CCNxName *name3= ccnxName_CreateFromCString(str);
```

Output: "ccnx:/Name=test/Name=MiISAg%3D%3D"

`ccnx:/name=done`