NDN Technical Memo: Naming Conventions

NDN Project Team

Revision history

Revision	Revision date	Description
1	July 21, 2014	Initial release

1 Introduction

This document describes the NDN naming conventions that gradually evolved during the past years of NDN application development practice. We intend to revise and extends this document and the described conventions as we gain more understanding about requirements of NDN applications and their interactions with the components of the NDN architecture.

The aim of this document is to provide general guidelines for NDN application developers to design their application namespaces. The conventions described in this document are just RECOMMENDATIONS, and it is up to the application developers to decide whether to follow these conventions or to override them with their own definitions. However, following the well-established conventions allows design of applications that are most efficient for NDN infrastructure to handle, less painful to debug, best understandable to end-users, and most interoperable with the other applications that follow the same conventions.

The conventions developed in this document are inspired by the CCNx Basic Naming Conventions specification [2]. We added several changes to reflect our current understanding about the naming conventions based on the experience from application development.

2 Value of NDN Name Components

In the wire format, name components are encoded as pure byte strings encapsulated in TLV-style headers and therefore it is possible to put any byte value in the component. However, NDN naming convention encourages the use of human-readable clear-text strings as name components, which resembles the file system naming scheme. UTF-8 encoding scheme is used to convert human-readable strings into byte representation in the packet. This allows accommodation to multiple languages in an internationalized environment.

For example, the name `/ndn/edu/ucla/melnitz/data/3/electrical/aggregate/power/instant` used in NDN Building Management System encodes the building name and sensor data type in clear text, which is intuitive and straightforward for the application end-users, in this case, the building managers.

The following list shows several cases where human-readable names are not feasible.

- Component encodes version and segment numbers with special markers, which is described later.
- Component encapsulates application-defined TLV component. For example, Signed Interest [4] requires embedding `SignatureInfo` and `SignatureValue` NDN TLV [3] components as part of the name:

  `/signed/interest/name/.../<SignatureInfo>/<SignatureValue>`

- Component encodes raw application specific data, such as cryptographic digests. For example,

  `/.../%00%C3...%BC%16`

Refer to NDN TLV specification [3] that defines the "NDN URI Scheme" with the specific conversion rules between human readable URI and wire (NDN TLV) format representations.

3 Functional Name Component: Special Markers

In a variety of situations, name components need to represent commands or annotations that can appear at arbitrary levels in the name hierarchy, intermixed with the human readable components that are analogous to pieces of file names. In these cases, the position in the name is not able to distinguish the components role and hence one must rely on coding conventions in the component itself. Wherever possible, however, the function or meaning of a name component should be determined by its context within the overall name and under the specific application semantics, and human readable components should be used where they will not be mistaken for functional name components.

The specification of UTF-8 prohibits certain octet values from occurring anywhere in a UTF-8 encoded string. These are the (hex) octet values `0xC0`, `0xC1`, and `0xF5` to `0xFF`. In addition, the null octet `00` is used for string termination and so does not encode a usable character. Thus, these make good *markers* to identify components of a NDN name that play special functional roles such as for versioning and segment numbering.

The following sections introduce some naming conventions that assign the prohibited octet values to the specific functions: segmenting, versioning, and sequencing. Note that the meaning of the special markers are application-layer information. From the router's perspective these functional components are no different from other name components. Although, the specific semantics of the special markers is ultimately defined by the application, the consistent use the naming convention helps applications to get the most advantage of the NDN architecture. For example, when the segmenting convention is properly used by the application, this application can get additional benefits from NDN routers, if a caching policy that prioritizes caching of the whole Data is in place. Further discussion about the specific properties and advantages that naming conventions, defining semantics of the name component(s), bring to the applications is outside the scope of this document. However, it is an active discussion within NDN project team and beyond.

3.1 Marker and Value Encoding

In the current convention, a functional name component is constructed using a *marker octet* from the UTF-8 prohibited set as the first octet, followed by the value in the following format based on non-negative integer encoding of NDN-TLV [3]:

```
NameComponentWithMarkerAndNumber ::= NAME-COMPONENT-TYPE TLV-LEGTH
                                       Marker
                                       includedNonNegativeInteger

Marker ::= BYTE

includedNonNegativeInteger ::= BYTE+

NDN-TLV := TLV-TYPE TLV-LENGTH TLV-VALUE?
TLV-TYPE := VAR-NUMBER
TLV-LENGTH := VAR-NUMBER
TLV-VALUE := BYTE+
```

The only difference between `includedNonNegativeInteger` and `nonNegativeInteger` defined in NDN-TLV spec [3] is the way the size of the stored non-negative integer is calculated. More specifically, length value of the TLV element containing `includedNonNegativeInteger` MUST be either 1, 2, 4, or 8 PLUS size of the prefixed elements. For `NameComponentWithMarkerAndNumber`, given that prefix is always a marker with size 1, the only values allowed values fro TLV-LENGTH are: 2, 3, 5, and 9.

Depending on the length MINUS size of the prefixed elements, a `includedNonNegativeInteger` is encoded as follows:

- if the length MINUS size of the prefixed elements is 1 (i.e., the value length is 1 octet), the `includedNonNegativeInteger` is encoded in one octet;

- if the length MINUS size of the prefixed elements is 2 (= value length is 2 octets), the `includedNonNegativeInteger` is encoded in 2 octets, in net byte-order;
- if the length MINUS size of the prefixed elements is 4 (= value length is 4 octets), the `includedNonNegativeInteger` is encoded in 4 octets, in net byte-order;
- if the length MINUS size of the prefixed elements is 8 (= value length is 8 octets), the `includedNonNegativeInteger` is encoded in 8 octets, in net byte-order.

In the future, we plan to extend the naming conventions and encoding methods to include different types of markers that could extend beyond a single naming component.

3.2 Segmenting

Segmenting is used when a large data is cut into several pieces. Each individual segment is assigned its own unique name and can be individually retrieved. Although not defined in this document, applications should strive to define segmentation in the way that makes possible to process individual segments on their own, without requiring waiting until the whole data is assembled. For example, a large video frame could be split into segments, each individually contributing complete pieces of the frame encoding.

For segmenting purposes, we define use of `NameComponentWithMarkerAndNumber` (Section 3.1) name component and two special markers (see Table 1): `0x00` for sequence-based segmentation and `0xFB` for byte offset segmentation. When sequence-based segmentation is used, each segmented Data packet should be assigned a sequential segment number, starting from 0: 0, 1, 2, ... When byte-offset segmentation is used, each segmented Data packet should be assigned a byte offset of the segmented content: 0, (content size in first Data packet), (sum of content sizes in first two Data packets), ...

The segment number is recommended to be put as the last name component, not including the implicit digest.

Table 1: Segmenting markers

Marker	Value meaning
`0x00`	Segment-number
`0xFB`	Byte offset

3.3 Versioning

In NDN, all Data packets are immutable and cannot be changed after being published. Therefore, when an applications needs to publish a "new" version of the data, it should be published under a new unique name using a proper versioning protocol. The main goal and property of the versioning protocol is to explicitly define the name-based ordering of Data versions: the "larger" is the name (in NDN-TLV canonical ordering [3]), the "newer" is the version of the Data.[1] This property can be used by third-parties, e.g., to prioritize caching of the latest versions of data.

The current convention define use of `NameComponentWithMarkerAndNumber` (Section 3.1) and `0xFD` marker to identify version name component (Table 2). The specific value for the version is determined by the applications and it is responsibility of the application to ensure that the larger value is assigned to the later version of the Data.

A simple way to implement version property in many environments is to use timestamp (the number of micro- or milliseconds since UNIX epoch) as a version value. Assuming data producers will take care of properly synchronizing their clocks, it is trivial to assign the correct version number in any multi-producer application without the need of introducing any version synchronization protocols. If the clock synchronization is not feasible, the application should use non-time-based versioning and/or make all attempts to ensure versioning property in some other way. Otherwise, the use of the defined versioning marker may harm the

[1] Without the use of versioning protocol, the implicit digest name component still makes all Data packets unique, but it is impossible to determine based on the name which version is "newer".

application, as third-parties may incorrectly treat Data packet with the "largest" version as the "latest" (e.g., incorrectly prioritize caching of older version).

Table 2: Versioning marker

Marker	Value meaning
`0xFD`	Non-negative integer (larger value for later version)

3.4 Time stamping

Some NDN applications may need to explicitly define when specific Data packet was produced. Even without synchronized clocks, this value can be used by third-parties to infer difference between generation of the Data packets by the same producer, and this knowledge can be used in some way (e.g., caches that try to keep time-diverse Data packets/measurements).

`NameComponentWithMarkerAndNumber` (Section 3.1) and the marker `0xFC` should be used when adding timestamp to the name (Table 3). The following value should be the number of microseconds since UNIX epoch (Thursday, 1 January 1970) not counting leap seconds, generated from the system clock to indicate the creation time of the Data.

Table 3: Timestamp marker

Marker	Value meaning
`0xFC`	Number of microseconds since UNIX epoch

3.5 Sequencing

Sequential data collections is another common feature in many NDN applications. Named Data Networking daemon (NFD) [1] uses such collections as part of the face status notification protocol; ChronoSync [5] protocol provide efficient synchronization primitives for sequenced data collections. Each item in these collections is numbered using monotonically increasing non-negative integer starting from zero. Collection items should be complete application-specific data items with or without application dependency on other collection items.

The following property, that can be leveraged by third-party applications, should hold for the items in the sequential data collections. If it is known that there exists a collection item with the sequence number X, then it is implied that items $0, 1, \ldots (X-1)$ has been already produced and item $(X+1)$ exists or could be produced in the future. (Note that this property does not require or guarantee that all "previous" items can be retrieved.)

Sequence number should be encoded as `NameComponentWithMarkerAndNumber` (Section 3.1) name component with marker `0xFE` followed by the sequence number (monotonically increasing non-negative integer) of the data item in the dataset (Table 4). Each data item is either an individual Data packet or a collection of multiple segmented Data packets.

Table 4: Sequencing marker

Marker	Value meaning
`0xFE`	Sequence number of the Data in the dataset

References

[1] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, et al. NFD developer's guide. Technical Report NDN-0021, NDN, July 2014.

[2] CCNx Project. CCNx basic name conventions. Online: http://www.ccnx.org/releases/latest/doc/technical/NameConventions.html, 2013.

[3] NDN Project Team. NDN packet format specification (version 0.1.1). Online: http://named-data.net/doc/ndn-tlv/, 2014.

[4] NFD Team. Signed Interest. Online: http://redmine.named-data.net/projects/ndn-cxx/wiki/SignedInterest, 2014.

[5] Zhenkai Zhu and Alexander Afanasyev. Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013)*, Goettingen, Germany, October 2013.