NDN Technical Memo: Naming Conventions

NDN Project Team

Revision history

Revision	Revision date	Description
2	October 03, 2019	Use typed name components instead of markers
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 [5] requires embedding `SignatureInfo` and `SignatureValue` TLV elements 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 Packet Format specification [4] 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: Typed Components

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 component's 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 NDN Packet Format specification, since version 0.3 [4], allows 65535 different TLV-TYPEs in name components. Several TLV-TYPEs have been reserved for functional name components, in accordance with the Name Component Assignment policy [3]. The following sections define the recommended semantics of these specific functions: segmenting, versioning, time stamping, and sequencing. Note that the meaning of these name component types is 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 name components is ultimately defined by the application, the consistent use of naming conventions helps applications get the most advantage from the NDN architecture. For example, proper use of segmenting convention can be beneficial to applications, if NDN routers prioritize caching of "complete" collections of Data packets.

3.1 Sequencing

Sequential data collections are a common feature in many NDN applications. The NDN Forwarding Daemon (NFD) [1] uses such collections as part of the face status notification protocol; the ChronoSync [6] protocol provides efficient synchronization primitives for sequenced data collections. Each item in these collections is numbered using monotonically increasing non-negative integers starting from zero. Collection items should be complete application-specific data items, and may or may not depend on other items of the same collection.

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

```
SequenceNumNameComponent = SEQUENCE-NUM-NAME-COMPONENT-TYPE TLV-LENGTH
                               nonNegativeInteger

SEQUENCE-NUM-NAME-COMPONENT-TYPE = %x25
```

A sequence number should be encoded as `SequenceNumNameComponent`, with the sequence number (monotonically increasing non-negative integer) of the data item in the collection as its TLV-VALUE. Each data item is either an individual Data packet or a collection of multiple segmented Data packets.

3.2 Segmenting

Segmenting is used when a large piece of data is cut into several chunks (or segments). 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 a way that makes it possible to process individual segments on their own, without requiring to wait 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.

```
SegmentNameComponent = SEGMENT-NAME-COMPONENT-TYPE TLV-LENGTH
                         nonNegativeInteger

ByteOffsetNameComponent = BYTE-OFFSET-NAME-COMPONENT-TYPE TLV-LENGTH
                            nonNegativeInteger

SEGMENT-NAME-COMPONENT-TYPE = %x21
BYTE-OFFSET-NAME-COMPONENT-TYPE = %x22
```

For segmenting purposes, we define `SegmentNameComponent` for sequence-based segmentation and `ByteOffsetNameComponent` for byte-offset segmentation. When sequence-based segmentation is used, each segment should be assigned a monotonically-increasing zero-based integer number: 0, 1, 2, ... The segment number is encoded in the TLV-VALUE of `SegmentNameComponent`. When byte-offset segmentation is used, each segment should be assigned a byte offset (number of bytes) from the beginning of the segmented content: 0, (content size in first segment), (sum of content sizes in first two segments), ... The segment number is encoded in the TLV-VALUE of `ByteOffsetNameComponent`.

It is recommended to put `SegmentNameComponent` and `ByteOffsetNameComponent` as the last explicit name component.

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 content, 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" the name is (in the canonical ordering [4]), the more recent the Data is.[1] This property can be used by third-parties, e.g., to prioritize caching of the latest version of the data.

```
VersionNameComponent = VERSION-NAME-COMPONENT-TYPE TLV-LENGTH
                         nonNegativeInteger

VERSION-NAME-COMPONENT-TYPE = %x23
```

`VersionNameComponent` contains a version number. The specific value 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.

Timestamps are commonly used as the version value. If using timestamps, it is recommended to use the number of microseconds since Unix epoch. In a multi-producer application, if the clocks are synchronized, it is trivial to assign non-decreasing version numbers without introducing a version synchronization protocol. If accurate clock synchronization is not feasible, the application should use non-time-based versioning and/or make all attempts to ensure proper versioning in some other way. Otherwise, the incorrect use of `VersionNameComponent` may be harmful to the application, as third-parties may incorrectly treat the Data packet with the "largest" version as the latest (e.g., incorrectly prioritize caching of older versions).

It is recommended to put `VersionNameComponent` just before `SegmentNameComponent` or `ByteOffsetNameComponent` in segmented content, or as the last explicit name component in non-segmented content.

3.4 Time Stamping

Some NDN applications may need to explicitly define when a specific Data packet was produced. Even without synchronized clocks, this value can be used by third-parties to infer the gap between the creation

[1]Without using a versioning protocol, the implicit digest name component still makes all Data packets unique, but it is impossible to determine the most recent version based on the naming alone.

times of different 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).

```
TimestampNameComponent = TIMESTAMP-NAME-COMPONENT-TYPE TLV-LENGTH
                           nonNegativeInteger

TIMESTAMP-NAME-COMPONENT-TYPE = %x24
```

`TimestampNameComponent` should be used when adding a timestamp to the name. The TLV-VALUE should be the number of microseconds since Unix epoch, not counting leap seconds, generated from the system clock to indicate the creation time of the Data.

References

[1] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, et al. NFD developer's guide. Technical Report NDN-0021, NDN, July 2018.

[2] CCNx Project. CCNx basic name conventions. Online: https://web.archive.org/web/20141020194539/http://www.ccnx.org/releases/latest/doc/technical/NameConventions.html, 2013.

[3] NDN Project Team. Name component type assignment. Online: https://redmine.named-data.net/projects/ndn-tlv/wiki/NameComponentType, 2018.

[4] NDN Project Team. NDN packet format specification (version 0.3). Online: https://named-data.net/doc/NDN-packet-spec/0.3/, 2018.

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

[6] 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.