Extending the Nexus Data Exchange Format (NDEF) Specification

Lawrence Fyfe
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4
Canada
ljfyfe@ucalgary.ca

Adam Tindale
OCAD University
100 McCaul Street
Toronto, ON M5T 1W1
Canada
atindale@faculty.ocadu.ca

Sheelagh Carpendale
University of Calgary
2500 University Drive NW
Calgary, AB T2N 1N4
Canada
sheelagh@ucalgary.ca

ABSTRACT

The Nexus Data Exchange Format (NDEF) is an Open Sound Control (OSC) namespace specification designed to make connection and message management tasks easier for OSC-based networked performance systems. New extensions to the NDEF namespace improve both connection and message management between OSC client and server nodes. Connection management between nodes now features human-readable labels for connections and a new message exchange for pinging connections to determine their status. Message management now has improved namespace synchronization via a message count exchange and by the ability to add, remove, and replace messages on connected nodes.

Keywords

OSC, namespace, specification, node, message, management, networking

1. INTRODUCTION

When musicians can spend less time on the configuration and setup of their networked performance systems, it is obvious that they can then spend more time on their actual performances. Yet it is often the case that the setup of computer music performance systems ends up taking time away from what performers should focus on, rehearsing and performing their actual music. This can be especially acute in the setup of laptop orchestras with large numbers of performers or in internet-based network performances, to name just two scenarios. Even when the technology works flawlessly, both scenarios often require so much time for setup that rehearsals are either rushed or do not happen at all. In general, networked performance systems could benefit from tools that make setup easier.

One way to make the setup of networked performance systems easier is to lower barriers to communication between systems. The MIDI protocol specification, since its introduction, has been both widely used and occasionally criticized [3]. The limitations of MIDI are beyond the scope of this paper but it is certainly true that MIDI is still in common use. One of the reasons that MIDI is still around is that it is a fixed protocol that can be understood by many devices. This kind of fixed protocol is very powerful, allowing for considerable interoperability between devices. In effect, MIDI provides a lingua franca for music performance systems.

In contrast, Open Sound Control (OSC) [20] is completely flexible by design, allowing for a wide variety of uses. This kind of flexibility is also powerful and allows for messaging systems that go far beyond the limits of the MIDI protocol. However, that same flexibility means that OSC makes for a poor lingua franca when there are no standard OSC message sets. One way to work around this is to develop specified namespaces within OSC. While there is value to this approach, it tends to take OSC closer to MIDI, losing some of the flexibility so prized with OSC. Ideally, OSC should retain its flexibility while simultaneously providing a simple lingua franca for systems to exchange namespaces.

Balancing flexibility with fixed namespaces, we created the Nexus Data Exchange Format (NDEF) specification [6], a namespace that serves as both a means to manage nodes on network-based performance systems and as a query system that enables an exchange of namespaces among nodes.

2. RELATED WORK

Work related to NDEF can be divided into two types: 1) OSC namespace specifications and 2) OSC query systems.

2.1 Namespace Specifications

Many OSC namespaces have been proposed with each one attempting to address a particular niche that is important to the proposers. TUIO [9] is an example of a successful namespace specification for tracking touches and fiducial markers developed for the reacTable [8] project. The Gesture Description Interchange Format (GDIF) [7] is a proposal to create an OSC namespace for storing and exchanging musical gesture data. SpatDIF [13] has an OSC namespace (among other formats) for storing and sharing information about spatial audio scenes. The use of the word "format" in GDIF and SpatDif provided inspiration for the naming of NDEF. Both GDIF and SpatDif are part of the Jamoma [14] project.

This is not an exhaustive list of namespaces but the previous examples make it clear that defined namespaces are useful enough in the context of OSC-based systems that developers take the time to create and use them.

2.2 Query Systems

NDEF is similar to the OSC query system proposal by Schmeder and Wright [18]. The proposed OSC query system allowed an OSC client to query an OSC server's namespace by including a '/' character at the end of a message pattern. The OSC server then sends back a reply containing any sub-patterns of the provided pattern. The reply begins

NIME'14, June 30 – July 03, 2014, Goldsmiths, University of London, UK.

with the '#' character to distinguish the query system from standard OSC messages. The following is an example from the proposal in which the sub-patterns of /foo/bar are requested. The #reply contains the type tag, the pattern, and the sub-patterns.

```
→ /foo/bar/
← #reply (sss) '/foo/bar/', 'test1', 'test2'
```

An important difference between NDEF and the proposed OSC query system is that NDEF does not require any changes to basic OSC implementations. Since the '#' character in the OSC query proposal is not part of the OSC specification [19], the query systems cannot be used by implementations of the basic specification. In contrast, NDEF is simply a defined namespace using basic OSC messages that can be implemented by any system that adheres to the original specification.

In Place et al [15], the authors recommend a scheme to make OSC into a more object-oriented system with the use of the ':' character to distinguish object's member methods and properties from standard OSC methods. The first two of the following messages are from the authors' examples in which the first message sets the value for the gain and the second message sets the value for a member :/unit of /gain. In this system, /gain is an object with both methods and properties and /unit is a property.

```
/module/audio/gain 120
/module/audio/gain:/unit midi
```

Beyond making OSC more object-oriented, the authors suggest some standard methods, including a /namespace method for querying the namespace of objects. The following message queries the namespace associated with gain:

```
/module/audio/gain:/namespace
```

Malloch et al [10] describe a digital orchestra system that uses /namespace appended (without the ':') to the end of messages to query both controllers and synthesizers for their namespaces. Those messages can then be mapped via their libmapper [11] library which enables mapping as well as namespace queries.

While the notion of an object-oriented OSC is interesting and mapping tools are clearly useful, the goal of NDEF is to address a more specific problem: making the management of OSC-based networked performance systems easier. The limited scope of NDEF makes implementation more straightforward for people looking to solve this specific problem.

3. EXTENDING NDEF

Since NDEF is an OSC namespace specification, it can be extended as new functionality is needed. After providing some background about the design of NDEF, the following subsections describe the new extensions to the specification. Note that, in the extended message descriptions, types shown in parenthesis are optional. Parenthesis are not used in actual NDEF type tags, as per the basic OSC specification.

3.1 Background

The NDEF specification [5] fits the Object-oriented Programming (OOP) style [17] described by Schmeder, et al but with no special characters to distinguish objects and methods. Instead, the terms used in the namespace are either simple nouns or verbs, with nouns being objects (OSC containers) and verbs being (OSC) methods. The root container for all NDEF messages is /ndef and there are two objects under the root, /connection and /message.

The use of the object-oriented style in NDEF is to strongly emphasize the human readable nature of both OSC generally and NDEF specifically. The intention behind the creation of NDEF is to provide a means to help people manage their performance systems rather than to provide a full zero-configuration networking [2] system. In other words, NDEF is for people who want to have both ease of use while still retaining full control over their performance systems.

3.2 Connection Management

In NDEF, establishing a connection is really about identifying a node. The following NDEF extensions help with both node identification and with testing connections between nodes.

3.2.1 Labelling

All NDEF messages provide an IP address and port number (the "si" in all type tags) that identify the source of the message. As such, the minimal form of an NDEF message is:

```
/ndef/[container]/[method] [IP address] [port]
```

This helps clients and servers identify each other. However, IP addresses and port numbers might be harder to remember for people looking at that information. Being able to identify a node with a human-readable name can be helpful. This is the basis for the existence of the DNS system [12]. In most circumstances, using DNS in networked music performances is not ideal. To allow for human-readable names without having to use DNS, NDEF connection messages now have an optional label string. Labels can be particularly useful when several nodes are present such as in laptop orchestras.

A label can be set when a connection is requested by adding an optional string argument to the request:

```
/ndef/connection/request,si(s)
```

The following is an example using the string "laptop1" as a label for the node at address 192.168.1.1 and port 7000:

```
/ndef/connection/request, "192.168.1.1" 7000
"laptop1"
```

Labels can also be set on connection replies:

```
/ndef/connection/accept,si(s)
```

Including the label string in the connection /request or /accept message makes it easy to set labels during node setup. In other situations, it may be necessary to change the name of a connection label after a connection has been established. In order to facilitate that, a new /label command is part of the specification:

```
/ndef/connection/label,sis
```

The last argument in the /label command is the label string to set on the receiver as with the "laptop1" example label described earlier. When a label has been accepted, the responder can send a /mark message to the requester. The term mark is used for this message in both the sense of taking notice and as another word for a label.

```
/ndef/connection/mark,si(s)
```

The mark message has an optional string for a label to return to the requester in case the responding node already has a label and wants to provide that to the requester. The handling of label conflicts is implementation dependent to allow for flexibility.

3.2.2 Pinging

The way that connections are handled via NDEF is conceptually similar to the three-way handshake used in TCP [1]. However, unlike TCP, after an NDEF connection is established, a node may become unavailable while still being "connected". Once a connection is established, it is useful to be able to test that connection. To allow for connection testing, NDEF now has a /ping command:

```
/ndef/connection/ping,si
```

If the node receiving the /ping message is available, it will send an /echo message to the originating node:

```
/ndef/connection/echo,si
```

NDEF ping messaging is similar to the ICMP protocol [16] used by the ping utility available with most operating systems. While an ICMP ping is useful for determining the general availability of a node on the network, it cannot determine when both OSC messaging and NDEF messaging are available. A node might be available on the network without either service running.

The /ping and /echo exchange can be used to determine both availability and round-trip time (latency) though timing is entirely implementation dependent. NDEF implementations also are free to determine how frequently /ping messages are sent and what sort of time-out mechanism is in place when /echo messages have not been received.

The /ping and /echo exchange is useful, for example, in a laptop orchestra. Often it is not always clear that nodes (laptops) in the orchestra are actually receiving the intended OSC messages. With the NDEF /ping and /echo, laptops could be periodically polled to determine whether they are actually receiving OSC messages, making it easier to identify nodes that are not receiving messages.

3.3 Message Management

Using NDEF, nodes can exchange messages in their OSC namespace, allowing namespaces to be synchronized.

3.3.1 Counting

When exchanging messages between nodes, it can be useful to know the number of messages that should be exchanged. By getting the number of messages, a node can know whether it has all of the messages contained in another node. NDEF now has a /count command for getting the number of messages from a remote node:

```
/ndef/message/count,si
```

The receiving node can then send a /tally message with the number of messages it contains. The last integer argument in /tally is the message count.

```
/ndef/message/tally,sii
```

With a /count and /tally exchange, a node can check the message count from another node to validate the number of messages it has received from a message request and to resend the request if necessary.

3.3.2 Editing

A set of new message editing extensions allow nodes to go beyond the exchanging of namespaces. The new extensions enable nodes to add, remove and replace the messages of other nodes, allowing for namespace synchronization. By synchronizing namespaces with the extended NDEF, the task of having OSC clients and servers share a common namespace is made easier.

The new /add message allows a node to add a message to another node.

```
/ndef/message/add,sis
```

The following example adds a new message called "/foo/bar" to the receiving node's namesapce:

```
/ndef/message/add, "192.168.1.1" 7000 "/foo/bar"
```

Each /add contains a single message but multiple /add messages can build an entire namespace on another node.

Messages can also be removed from a node with the new /remove message:

```
/ndef/message/remove,sis
```

This message removes "/foo/bar" from the receiving node:

```
/ndef/message/remove, "192.168.1.1" 7000
"/foo/bar"
```

The new add and remove extensions having the following basic structure:

```
/ndef/message/[method] [IP address] [port] [OSC
message]
```

The OSC message argument should include the address pattern and the type tag, formatted as a standard OSC message. Arguments should not be included.

A message can be replaced with another message using /replace:

```
/ndef/message/replace,siss
```

The following /replace command replaces "/foo/bar" with "/bar/foo":

```
/ndef/message/replace, "192.168.1.1" 7000
"/foo/bar" "/bar/foo"
```

The replace extension has four arguments with the last two arguments being the old message string and the new message string that will replace it. The general structure of replace messages:

```
/ndef/message/replace [IP address] [port] [old
message] [new message]
```

Of the three new messages, the /replace message is the most powerful since the fact that it replaces a message means that the mapping that used the old message could still be valid with the new message. By retaining the mapping, the OSC client could simply send the new message without having to know the details of the mapping on the server.

The /add, /remove, and /replace messages are not just meant to be sent to OSC servers that contain mappings. They can also be sent from OSC servers to clients used for control. This allows for two-way namespace synchronization.

3.4 Acknowledging Requests

Most of the methods associated with the two containers, /connection and /message, come in pairs, a requesting method and an acknowledging method. The names used for each pair reflect their purpose as well as delineating their status as a request or an acknowledgement. Table 1 summarizes the pairs.

None of the editing messages has an equivalent acknowledging message. Instead, when a requester makes edits to messages on a receiving node, the requester can simply make a new message request to determine whether the new messages have been accepted by the receiving node.

Container	Requesting	Acknowledging
/connection	/request /ping /label	/accept /echo /mark
/message	/request /count	/reply /tally

Table 1: A summary of requesting and acknowledging methods.

3.5 Implementation

The JunctionBox [4] interaction toolkit features the first full implementation of NDEF. Since JunctionBox is open source software, developers wanting to implement NDEF can use the JunctionBox code as a template.

4. CONCLUSIONS

Extending the capabilities of the NDEF specification makes it more useful for the setup of OSC-based networked performance systems. The following setup-related issues have been specifically addressed in the new NDEF extensions:

- The specification now includes the ability to label connections with a human-readable string.
- New ping messages allow a node to be pinged about the status of both OSC and NDEF.
- Nodes can now be queried about the number of message that they contain, allowing nodes to determine whether they have exchanged all messages and to send a new request if the message counts do not match.
- Message management among nodes is made easier by the inclusion of new editing messages that enable nodes to add, remove and replace messages on other nodes, allowing for namespace synchronization.

5. ACKNOWLEDGEMENTS

We would like to thank the University of Calgary, OCAD University, NSERC, GRAND, SurfNet, AITF, and SMART Technologies for research support.

6. REFERENCES

[1] V. Cerf, Y. Dalal, and C. Sunshine. Specification of internet transmission control program. `http://tools.ietf.org/html/rfc675`, 1974.

[2] S. Cheshire. Zero configuration networking (zeroconf). `http://www.zeroconf.org/`, 2013.

[3] F. Richard Moore. The Dysfunctions of MIDI. *Computer Music Journal*, 12(1):19–28, 1988.

[4] L. Fyfe. JunctionBox. `http://innovis.cpsc.ucalgary.ca/Software/JunctionBox`, 2014.

[5] L. Fyfe. The Nexus Data Exchange Format Specification. `http://innovis.cpsc.ucalgary.ca/Research/NDEFSpecification`, 2014.

[6] L. Fyfe, A. Tindale, and S. Carpendale. Node and Message Management with the JunctionBox Interaction Toolkit. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 520–521, 2012.

[7] A. R. Jensenius, T. Kvifte, and R. I. Godøy. Towards a Gesture Description Interchange Format. In *Proceedings of the Conference on New Interfaces for Musical Expression*, pages 176–179. IRCAM–Centre Pompidou, 2006.

[8] S. Jordà, M. Kaltenbrunner, G. Geiger, and R. Bencina. The reacTable*. In *Proceedings of the International Computer Music Conference*, pages 579–582, 2005.

[9] M. Kaltenbrunner, T. Bovermann, R. Bencina, and E. Costanza. TUIO - A Protocol for Table Based Tangible User Interfaces. In *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, Vannes, France, 2005.

[10] J. Malloch, S. Sinclair, and M. M. Wanderley. From Controller to Sound: Tools for Collaborative Development of Digital Musical Instruments. In *Proceedings of the International Computer Music Conference*, 2007.

[11] J. Malloch, S. Sinclair, and M. M. Wanderley. Libmapper (A Library for Connecting Things). In *Proceedings of the International Conference on Human Factors in Computing Systems*, pages 3087–3090, 2013.

[12] P. Mockapetris. Domain Names - Concepts and Facilities. `http://tools.ietf.org/html/rfc1034`, 1987.

[13] N. Peters, T. Lossius, and J. C. Schacher. SpatDIF: Principles, Specification, and Examples. In *Proceedings of the 9th Sound and Music Computing Conference*, pages 500–505, 2012.

[14] T. Place and T. Lossius. Jamoma: A Modular Standard for Structuring Patches in Max. In *Proceedings of the International Computer Music Conference*, 2006.

[15] T. Place, T. Lossius, A. R. Jensenius, N. Peters, and P. Baltazar. Addressing Classes by Differentiating Values and Properties in OSC. In *Proceedings of the Conference on New Interfaces for Musical Expression*, 2008.

[16] J. Postel. Internet Control Message Protocol. `http://tools.ietf.org/html/rfc792`, 1981.

[17] A. Schmeder, A. Freed, and D. Wessel. Best Practices for Open Sound Control. In *Linux Audio Conference*, 2010.

[18] A. W. Schmeder and M. Wright. A Query System for Open Sound Control. In *OpenSoundControl Conference*, 2004.

[19] M. Wright. The Open Sound Control 1.0 Specification. `http://opensoundcontrol.org/spec-1_0`, 2002.

[20] M. Wright. Open Sound Control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.