A Query System for Open Sound Control

Draft Proposal

Andrew W. Schmeder, Matthew Wright
andyschm@cnmat.berkeley.edu, matt@cnmat.berkeley.edu

Center for New Music and Audio Technology, University of California at Berkeley
www.cnmat.berkeley.edu

15th July 2004

Abstract

A query system is proposed for inter-application control scenarios. The features consist of namespace exploration, documentation, type-signature, return-type-signature and parameter constraint specification, current-value polling, identification of common interpretation maps via osc-schema, as well as a general error reporting and reply mechanism.

1 Preamble

This document describes the query system for Open Sound Control, to be presented at the Open Sound Control Conference 2004 (CNMAT 2004). The reader should be familiar with the Open Sound Control 1.0 Specification (Wright 2002). This is a draft document. Comments should be directed to the authors and/or the osc-dev mailing list.

2 Introduction

The OSC protocol is characterized by the following: It is a stateless datagram protocol, is compatible with real-time systems (e.g., communication requires small and bounded resources), is lightweight (i.e., can be implemented with a small program), and uses a loosely-structured human-readable routing system (i.e., uses a hierarchical namespace).

OSC has enjoyed considerable success in a wide spectrum of scenarios, including integration into many popular audio applications/environments and programming languages (Wright, Freed, and Momeni 2003).

3 Motivation

Implementors of OSC-enabled systems have encountered requirements outside the scope of the OSC 1.0 Specification. From these real-world use-case scenarios, it is clear that the community has broader requirements that must be considered.

Herein we are especially interested in the problem of inter-application control as seen in multi-tier, collaborative, and distributed multimedia systems. In decentralized networks with multiple OSC-enabled systems, it is necessary to have a mechanism for discovery; essentially mapping out the namespaces of the actors to determine interconnection potential. The query system proposed is designed to meet this requirement, while at the same time retaining the original design goals of OSC.

4 The Query System

4.1 Definition

A query is a pair of messages; the *prompt*, a message transmitted from client to server, which encodes a question; and the *reply*, which is the answer to the request returned to the client. When no logical reply exists, the server may return an *error* instead.

4.2 Transport Requirements

The transport is responsible for providing the mechanism to deliver the reply to the prompt. For example, with UDP/IP simply invert the source/destintation IP addresses and ports; with TCP/IP, use a bi-directional connection. The transport is also responsible for discovery, or the ability to detect and locate OSC servers (there is no recommendation for what protocol to use at this time).

4.3 Query Message Format

The query system uses standard OSC messages, however the address patterns may contain the special control character '#', which is normally forbidden in OSC addresses. Query address patterns are compatible with existing features including pattern matching expressions and bundles, and do not reserve or restrict any part of the address space.

4.3.1 Prompt Message Format

The prompt message begins with a typical OSC address pattern and ends with the character '/' or the pattern '/#command' where command is replaced by the query type. Additional arguments may be present, depending on the query type.

4.3.2 Reply Message Format

Reply messages use the special address '#reply'. Reply messages must contain sufficient information to determine the prompt, to enable the client to process query replies asynchronously. Therefore, the first argument of a reply message is either the string containing the prompt address pattern, or a blob containing the entire prompt message. We recommend that implementations use the blob format when the prompt contains arguments. Additional arguments may be present depending on the scenario.

4.3.3 Error Message Format

The error message is a type of reply message using the special address '#error', and having as its second argument a short string which defines the error code (defined below), and an optional third argument giving a string documenting the error.

4.3.4 Handling Pattern Expansion

Use of address-pattern expressions with a query prompt is equivalent to sending the prompt to every complete matching address. The server's response should be as if this were the case.

```
→ /*/#documentation
← #reply (ss) '/patch1/#documentation', 'A patch.'
← #reply (ss) '/main_clock/#documentation', 'A clock.'
...
```

4.4 Error Codes

undefined Undefined response / query method not supported. In some situations a query has no logical answer; e.g., if no documentation exists, if no type-signature is accepted, etc. Server may return the 201 error code in this case.

```
→ /patch1/#documentation
← #error (sss) '/patch1/#documentation', 'undefined',
       '@todo: write me!'
```

relocated Redirection; the object has moved, return indicates a new target address. Third argument is the address of the new location; fourth is documentation. Support for this type of error is not required.

```
→ /patch1/osc1
← #error (ssss) '/patch1/osc1', 'relocated', '/patch1/osc2'
```

corrupt Indicates a parser error occured while the OSC engine was validating the packet. Could be caused by a bad address, bad type-tag string, incorrect packing of data, etc.

```
→ /patch1/osc#
← #error (sss) '/patch1/osc#', 'syntax', 'Invalid address.'
```

mismatch Invalid type-signature for destination (input does not match type-signature).

```
→ /patch1/osc1/phase (s) 'hello'
← #error (bss) [[ /patch1/osc1/phase (s) 'hello' ]], 'mismatch', 'Expecting a single float32.'
```

infeasable Input constraint error. Input types failed to meet application requirements.

```
→ /patch1/osc1/freq (f) -440.0
← #error (bss) [[ /patch1/osc1/freq (f) -440.0 ]], 'infeasable',
       'Negative frequencies not allowed.'
```

missing Address not found.

```
→ /ptach1/osc1
← '#error' (sss) '/ptach1/osc1' 'missing', 'Address non-existant.'
```

May also be used when a pattern-matching query matches zero complete addresses.

```
→ /patch1/osc*/amplitude (f) 0.0
← #error (bss) [[ /patch1/osc*/amplitude (f) 0.0 ]],
      'missing', 'No matches.'
```

failed Unexpected internal server/system error (generic message).

```
→ /patch1/copy-to (s) '/patch2'
← #reply (bss) [[ /patch1/copy-to ... ]], 'failed', 'Out of memory.'
```

4.4.1 Discussion

Error codes are useful in situations outside of the query system, and we leave that option open to implmentors.

4.5 Query Types

4.5.1 Namespace Exploration

Prompt to discover what addresses appear at the next level below a base address. Prompt address appears as a normal address pattern which ends with the '/' character.

```
→ /foo/bar/
← #reply (sss) '/foo/bar/', 'test1', 'test2'
→ /foo/bar/test1/
← #reply (s) '/foo/bar/test1/'  # is a terminal node.
```

4.5.2 Documentation

Obtains user documentation; returns one or more short text strings and/or URLs for obtaining documentation. Prompt address pattern ends with '/#documentation'.

```
→ /foo/bar/#documentation
← #reply (sss) '/foo/bar/#documentation' 'Application XYZ v0.1-beta',
        'http://myapp.devcenter.org'
```

Implementations may support documentation in multiple languages, in which case the client may specify a list of languages, in order of preference, as arguments to the prompt message. Language is specified according to RFC3066 (Alvestrand 2001), preferably using a 2 or 3 character language code. When the client uses a language specification, the server is required to specify the language as the second argument in the reply. If none of the requested languages are available, the reply may use a default language.

```
→ /patch1/osc1/phase/#documentation (s) 'fr', 'de'
← #reply (bss) [[ /patch1/osc1/phase/#documentation (s) 'fr', 'de' ]],
        'en', 'Phase in radians'
```

4.5.3 Type Signature

Returns a *type-tag string* describing the server's expected input message layout. OSC pattern-matching expressions are valid type-tag strings, as is the empty string.

```
→ /patch1/osc1/phase/#type-signature
← #reply (ss) '/patch1/osc1/phase/#type-signature' 'f'
...
→ /patch1/osc1/phase (f) 0.127
```

The type-signature may contain extra arguments which describe parameter constraints or other meta-data. We recommend doing so with the dictionary type. The following keys for numeric types are useful: units, min, max, step, interval-type. For string types: allow-chars (i.e., character class such as 'a-zA-Z0-9'), min-length, max-length, min and max (lexicographic bounds).

```
→ /patch1/osc1/phase/#type-signature
← #reply (ss{sssfsfss}) '/patch1/osc1/phase/#type-signature', 'f',
      { units = 'degrees',
        min = 0.0,
        max = 360.0,
        interval-type = 'half-open-upper' }
```

4.5.4 Current Value

For addresses which correspond to readable parameters in the server, this query enables the client to request the current value. Generally we expect that the message returned by current-value corresponds with the type-signature information.

```
→ /patch1/gain1 100.0
→ /patch1/gain1/#current-value
← #reply (sf) '/patch1/gain1/#current-value', 100.0
```

4.5.5 Return Type Signature

If the target address will generate a return message, e.g., typically when the address represents a function or method, this query returns a *type-tag string* describing the message layout that the client can expect. The same format rules apply as in the type-signature query.

```
→ /patch1/get-filename/#return-type-signature
← #reply (ss) '/patch1/osc1/export/#return-type-signature', 'f'
→ /patch1/get-filename
← #reply (ss) '/patch1/get-filename', '~/projects/testpatch1.xml'
```

4.5.6 OSC Schema

Use of OSC-schema is supported by the special method `/#osc-schema`, which returns a list of schema in-use at the corresponding base address. A full discussion of this feature is found in the next section.

5 OSC Schema - Namespace Interpretation Maps

The use of human-readable address format allows ad-hoc construction of the namespace. However, for some common conceptual structures, it is useful to have standard interpretation maps, i.e., consistent namespace structuring for particular applications, which may be compatible across diverse OSC-enabled systems.

Simple use-cases may include parameter definitions (e.g., knob, switch), human-computer interface device classes (e.g., joystick, tablet), 'OSC parsed MIDI' to/from raw MIDI. Advanced schema may include connection streaming (i.e., server-to-client event notifications), clock synchronization, distributed computing, and interfaces to object-modular environments.

We propose a *schema identification* system; a schema is identified by a unique URL. There is no formal registration process, and no requirements for the schema URL, other than that the domain name must be valid and we generally expect the author to post appropriate descriptive content there. Some schema may receive an *endorsement* from CNMAT researchers by allowing use of the opensoundcontrol.org domain for identification.

```
→ /patch1/midi-source/#osc-schema
← #reply (sss) '/patch1/midi-source/#osc-schema',
      'http://osw.sourceforge.net/docs/object.html',
      'http://www.opensoundcontrol.org/schemas/midi/source.html'
```

We invite users and implementors of OSC to consider, create, document and share useful schemas.

References

Alvestrand, H. (2001). RFC 3066: Tags for the identification of languages.

CNMAT (2004). Open sound control conference, 2004.

Wright, M. (2002). Open sound control specification version 1.0.

Wright, M., A. Freed, and A. Momeni (2003). Open sound control: State of the art 2003. In *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*.