Appendix A

OpenSound Control Specification

Version 1.0, March 26 2002, Matt Wright.
LaTeX by Uli Franke.

OpenSound Control (OSC) is an open, transport-independent, message-based protocol developed for communication among computers, sound synthesizers, and other multimedia devices.

A.1 OSC Syntax

This section defines the syntax of OSC data.

A.1.1 Atomic Data Types

All OSC data is composed of the following fundamental data types:

int32: 32-bit big-endian two's complement integer

OSC**-timetag:** 64-bit big-endian fixed-point time tag, semantics defined below

float32: 32-bit big-endian IEEE 754 floating point number

OSC**-string:** A sequence of **non-null** ASCII characters **followed by a null**, followed by 0-3 additional null characters to make the total number of bits a multiple of 32. (OSC-string examples) In this document, example OSC-strings will be written without the null characters, surrounded by double quotes.

OSC**-blob:** An int32 size count, followed by that many 8-bit bytes of arbitrary binary data, followed by 0-3 additional zero bytes to make the total number of bits a multiple of 32.

The size of every atomic data type in OSC is a multiple of 32 bits. This guarantees that if the beginning of a block of OSC data is 32-bit aligned, every number in the OSC data will be 32-bit aligned.

OSC Type Tag	Type of corresponding argument
i	int32
f	float32
s	OSC-string
b	OSC-blob

Table A.1: The meaning of each OSC Type Tag.

A.1.2 OSC Packets

The unit of transmission of OSC is an OSC Packet. Any application that sends OSC Packets is an OSC Client; any application that receives OSC Packets is an OSC Server. An OSC packet consists of its contents, a contiguous block of binary data, and its size, the number of 8-bit bytes that comprise the contents. The size of an OSC packet is always a multiple of 4.

The underlying network that delivers an OSC packet is responsible for delivering both the contents and the size to the OSC application. An OSC packet can be naturally represented by a datagram by a network protocol such as UDP. In a stream-based protocol such as TCP, the stream should begin with an int32 giving the size of the first packet, followed by the contents of the first packet, followed by the size of the second packet, etc.

The contents of an OSC packet must be either an OSC Message or an OSC Bundle. The first byte of the packet's contents unambiguously distinguishes between these two alternatives.

A.1.3 OSC Messages

An OSC message consists of an OSC Address Pattern followed by an OSC Type Tag String followed by zero or more OSC Arguments.

Note: Some older implementations of OSC may omit the OSC Type Tag string. Until all such implementations are updated, OSC implementations should be robust in the case of a missing OSC Type Tag String.

OSC Address Patterns: An *OSC Address Pattern* is an OSC-string beginning with the character '/' (forward slash).

OSC Type Tag String: An *OSC Type Tag String* is an OSC-string beginning with the character ',' (comma) followed by a sequence of characters corresponding exactly to the sequence of OSC Arguments in the given message. Each character after the comma is called an OSC Type Tag and represents the type of the corresponding OSC Argument. (The requirement for OSC Type Tag Strings to start with a comma makes it easier for the recipient of an OSC Message to determine whether that OSC Message is lacking an OSC Type Tag String.)

Table A.1 lists the correspondance between each OSC Type Tag and the type of its corresponding OSC Argument. Some OSC applications communicate among instances of themselves with additional, nonstandard argument types beyond those specified above. OSC applications are not required to recognize these types; an OSC application should discard any message whose OSC Type Tag String contains any unrecognized OSC Type

OSC Type Tag	Type of corresponding argument
h	64 bit big-endian two's complement integer
t	OSC-timetag
d	64 bit ("double") IEEE 754 floating point number
S	Alternate type represented as an OSC-string (for example, for systems that differentiate "symbols" from "strings")
c	an ascii character, sent as 32 bits
r	32 bit RGBA color
m	4 byte MIDI message. Bytes from MSB to LSB are: port id, status byte, data1, data2
T	True. No bytes are allocated in the argument data.
F	False. No bytes are allocated in the argument data.
N	Nil. No bytes are allocated in the argument data.
I	Infinitum. No bytes are allocated in the argument data.
[	Indicates the beginning of an array. The tags following are for data in the Array until a close brace tag is reached.
]	Indicates the end of an array.

Table A.2: OSC Type Tags that must be used for certain nonstandard argument types.

Tags. An application that does use any additional argument types must encode them with the OSC Type Tags in table A.1.

A.1.4 OSC Arguments

A sequence of OSC Arguments is represented by a contiguous sequence of the binary representations of each argument.

A.1.5 OSC Bundles

An OSC Bundle consists of the OSC-string "#bundle" followed by an OSC Time Tag, followed by zero or more OSC Bundle Elements. The OSC-timetag is a 64-bit fixed point time tag whose semantics are described below.

An OSC Bundle Element consists of its size and its contents. The size is an int32 representing the number of 8-bit bytes in the contents, and will always be a multiple of 4. The contents are either an OSC Message or an OSC Bundle.

Note this recursive definition: bundle may contain bundles.

A.2 OSC Semantics

This section defines the semantics of OSC data.

A.2.1 OSC Address Spaces and OSC Addresses

Every OSC server has a set of OSC Methods. OSC Methods are the potential destinations of OSC messages received by the OSC server and correspond to each of the points of control that the application makes available.

Data	Size	Purpose
OSC-string "#bundle"	8 bytes	How to know that this data is a bundle
OSC-timetag	8 bytes	Time tag that applies to the entire bundle
Size of first bundle element	int32 = 4 bytes	First bundle element
First bundle element's contents	As many bytes as given by "size of first bundle element"	First bundle element
Size of second bundle element	int32 = 4 bytes	Second bundle element
Second bundle element's contents	As many bytes as given by "size of second bundle element"	Second bundle element
etc.		Addtional bundle elements

Table A.3: This table shows the parts of a two-or-more-element OSC Bundle and the size (in 8-bit bytes) of each part.

"Invoking" an OSC method is analogous to a procedure call.

it means supplying the method with arguments and causing the method's effect to take place.

An OSC Server's OSC Methods are arranged in a tree strcuture called an OSC Address Space. The leaves of this tree are the OSC Methods and the branch nodes are called OSC Containers. An OSC Server's OSC Address Space can be dynamic; that is, its contents and shape can change over time.

Each OSC Method and each OSC Container other than the root of the tree has a symbolic name, an ASCII string consisting of printable characters other than the ones showed in table A.2.1.

The OSC Address of an OSC Method is a symbolic name giving the full path to the OSC Method in the OSC Address Space, starting from the root of the tree. An OSC Method's OSC Address begins with the character '/' (forward slash), followed by the names of all the containers, in order, along the path from the root of the tree to the OSC Method, separated by forward slash characters, followed by the name of the OSC Method. The syntax of OSC Addresses was chosen to match the syntax of URLs. (see OSC Address Examples)

A.2.2 OSC Message Dispatching and Pattern Matching

When an OSC server receives an OSC Message, it must invoke the appropriate OSC Methods in its OSC Address Space based on the OSC Message's OSC Address Pattern. This process is called dispatching the OSC Message to the OSC Methods that match its OSC Address Pattern. All the matching OSC Methods are invoked with the same argument data, namely, the OSC Arguments in the OSC Message.

The *parts* of an OSC Address or an OSC Address Pattern are the substrings between adjacent pairs of forward slash characters and the substring after the last forward slash

Char	Name	Code
' '	space	32
#	number sign	35
*	asterisk	42
,	comma	44
/	forward slash	47
?	question mark	63
[	open bracket	91
]	close bracket	93
{	open curly brace	123
}	close curly brace	125

Table A.4: Printable ASCII characters not allowed in names of `OSC` Methods or `OSC` Containers character name ASCII code (decimal)

character. (examples)

A received `OSC` Message must be dispatched to every `OSC` method in the current `OSC` Address Space whose `OSC` Address matches the `OSC` Message's `OSC` Address Pattern. An `OSC` Address Pattern matches an `OSC` Address if

1. The `OSC` Address and the `OSC` Address Pattern contain the same number of *parts*; and

2. Each part of the `OSC` Address Pattern *matches* the corresponding part of the `OSC` Address.

A part of an `OSC` Address Pattern matches a part of an `OSC` Address if every consecutive character in the `OSC` Address Pattern matches the next consecutive substring of the `OSC` Address and every character in the `OSC` Address is matched by something in the `OSC` Address Pattern. These are the matching rules for characters in the `OSC` Address Pattern:

1. "?" in the `OSC` Address Pattern matches any single character

2. "*" in the `OSC` Address Pattern matches any sequence of zero or more characters

3. A string of characters in square brackets (e.g., "`[string]`") in the `OSC` Address Pattern matches any character in the string. Inside square brackets, the minus sign (-) and exclamation point (!) have special meanings:

   - two characters separated by a minus sign indicate the *range* of characters between the given two in ASCII collating sequence. (A minus sign at the end of the string has no special meaning.)
   - An exclamation point at the beginning of a bracketed string negates the sense of the list, meaning that the list matches any character not in the list. (An exclamation point anywhere besides the first character after the open bracket has no special meaning.)

4. A comma-separated list of strings enclosed in curly braces (e.g., "`{foo,bar}`") in the `OSC` Address Pattern matches any of the strings in the list.

5. Any other character in an `OSC` Address Pattern can match only the same character.

A.2.3 Temporal Semantics and OSC Time Tags

An OSC server must have access to a representation of the correct current absolute time. OSC does not provide any mechanism for clock synchronization.

When a received OSC Packet contains only a single OSC Message, the OSC Server should invoke the correponding OSC Methods immediately, i.e., as soon as possible after receipt of the packet. Otherwise a received OSC Packet contains an OSC Bundle, in which case the OSC Bundle's OSC Time Tag determines when the OSC Bundle's OSC Messages' corresponding OSC Methods should be invoked. If the time represented by the OSC Time Tag is before or equal to the current time, the OSC Server should invoke the methods immediately (unless the user has configured the OSC Server to discard messages that arrive too late). Otherwise the OSC Time Tag represents a time in the future, and the OSC server must store the OSC Bundle until the specified time and then invoke the appropriate OSC Methods.

Time tags are represented by a 64 bit fixed point number. The first 32 bits specify the number of seconds since midnight on January 1, 1900, and the last 32 bits specify fractional parts of a second to a precision of about 200 picoseconds. This is the representation used by Internet NTP timestamps.The time tag value consisting of 63 zero bits followed by a one in the least signifigant bit is a special case meaning ”immediately.”

OSC Messages in the same OSC Bundle are atomic; their corresponding OSC Methods should be invoked in immediate succession as if no other processing took place between the OSC Method invocations.

When an OSC Address Pattern is dispatched to multiple OSC Methods, the order in which the matching OSC Methods are invoked is unspecified. When an OSC Bundle contains multiple OSC Messages, the sets of OSC Methods corresponding to the OSC Messages must be invoked in the same order as the OSC Messages appear in the packet. (example)

When bundles contain other bundles, the OSC Time Tag of the enclosed bundle must be greater than or equal to the OSC Time Tag of the enclosing bundle. The atomicity requirement for OSC Messages in the same OSC Bundle does not apply to OSC Bundles within an OSC Bundle.

A.3 OpenSound Control Spec Examples

This section provides examples to support the OpenSound Control Specification. It was taken from *OpenSound Control Spec Examples* by Matt Wright, Version 1.0, March 29 2002.

A.3.1 OSC-string examples

The string ”OSC” is represented as an OSC-string with these four bytes:

```
O S C \0
```

The string ”data” is represented as an OSC-string with these eight bytes:

```
d a t a \0 \0 \0 \0
```

A.3.2 OSC Type Tag String Examples

See table A.3.2.

Argument types	`OSC` Type Tag String
One float32 argument	”`,f`”
Two int32 arguments followed by one `OSC`-string argument followed by three float32 arguments	”`,iisfff`”
No arguments	”`,`”
An int32 argument followed by two `OSC`-blob arguments	,ibb

Table A.5: Example `OSC` Type Tag Strings

A.3.3 `OSC` Address Examples

Suppose a particular `OSC` Address Space includes an `OSC` Method with the name ”`frequency`”. This method is contained in an `OSC` Container with the name ”`3`”, which is contained in another `OSC` container named ”`resonators`”, which is contained in the `OSC` container that is the root of the address space tree. The method’s `OSC` Address is ”`/resonators/3/frequency`”.

The `OSC` Address ”`/a/b/c/d/e`” means that:

- The root of the tree contains an `OSC` Container with the name ”`a`”,
- that `OSC` Container contains an `OSC` Container with the name ”`b`”,
- that `OSC` Container contains an `OSC` Container with the name ”`c`”,
- that `OSC` Container contains an `OSC` Container with the name ”`d`”, and
- that `OSC` Container contains an `OSC` Method with the name ”`e`”.

A.3.4 `OSC` Address Parts Examples

There are three parts of the `OSC` Address ”`/a/b/cde`”: ”`a`”, ”`b`”, and ”`cde`”. Note that the last part is the name of the `OSC` Method and the other parts are the names of the `OSC` Containers that (recursively) contain the method.

There are three parts of the `OSC` Address pattern ”`/?/b/*c`”: ”`?`”, ”`b`”, and ”`*c`”.

A.3.5 `OSC` Message Examples

In each of these examples, each byte of a message is printed first in hexadecimal, followed by the corresponding ASCII character in parentheses.

The `OSC` Message with the `OSC` Address Pattern ”`/oscillator/4/frequency`” and the floating point number 440.0 as the single argument would be represented by the following 32-byte message:

```
2f (/)  6f (o)  73 (s)  63 (c)
69 (i)  6c (l)  6c (l)  61 (a)
74 (t)  6f (o)  72 (r)  2f (/)
34 (4)  2f (/)  66 (f)  72 (r)
65 (e)  71 (q)  75 (u)  65 (e)
```

```
6e (n)  63 (c)  79 (y)  0  ( )
2c (,)  66 (f)  0  ( )  0  ( )
43 (C)  dc (Ü)  0  ( )  0  ( )
```

The next example shows the 40 bytes in the representation of the OSC Message with OSC Address Pattern "/foo" and 5 arguments:

1. The int32 1000
2. The int32 -1
3. The string "hello"
4. The float32 1.234
5. The float32 5.678

```
2f (/)  66 (f)  6f (o)  6f (o)
0  ( )  0  ( )  0  ( )  0  ( )
2c (,)  69 (i)  69 (i)  73 (s)
66 (f)  66 (f)  0  ( )  0  ( )
0  ( )  0  ( )  3  ( )  e8 (è)
ff (ß)  ff (ß)  ff (ß)  ff (ß)
68 (h)  65 (e)  6c (l)  6c (l)
6f (o)  0  ( )  0  ( )  0  ( )
3f (?)  9d ( )  f3 (ó)  b6 (■)
40 (@)  b5 (ţ)  b2 (Ť)  2d (-)
```

A.3.6 Order of Invocation of OSC Methods matched by OSC Messages in an OSC Bundle

Suppose an OSC Servers' OSC Address Space includes methods with the following OSC Addresses:

- "/first/this/one"
- "/second/1"
- "/second/2"
- "/third/a"
- "/third/b"
- "/third/c"

Suppose an OSC Bundle is received that contains three OSC Messages, and that the three OSC Messages have these OSC Address Patterns:

1. /first/this/one
2. /second/[1-2]
3. /third/*

Six methods will be invoked in this order:

(1) First ”/first/this/one”, since that OSC Address Pattern appeared first in the OSC Bundle; then

(2-3) Either ”/second/1” followed ”/second/2” or ”/second/2” followed by ”/second/1”; then

(3-6) ”/third/a”, ”/third/b”, and ”/third/c”, in any order.

There are twelve possible orders in which an OSC server may invoke these six methods:

1. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/a”, ”/third/b”, ”/third/c”
2. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/a”, ”/third/c”, ”/third/b”
3. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/b”, ”/third/a”, ”/third/c”
4. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/b”, ”/third/c”, ”/third/a”
5. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/c”, ”/third/a”, ”/third/b”
6. ”/first/this/one”, ”/second/1”, ”/second/2”, ”/third/c”, ”/third/b”, ”/third/a”
7. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/a”, ”/third/b”, ”/third/c”
8. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/a”, ”/third/c”, ”/third/b”
9. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/b”, ”/third/a”, ”/third/c”
10. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/b”, ”/third/c”, ”/third/a”
11. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/c”, ”/third/a”, ”/third/b”
12. ”/first/this/one”, ”/second/2”, ”/second/1”, ”/third/c”, ”/third/b”, ”/third/a”