

# **MAL**

MIDI Abstraction Layer

Version 0.2

[www.silverblade.co.uk](http://www.silverblade.co.uk)

© Andrew Greenwood 2007

## Welcome to the MIDI Abstraction Layer

Dealing with MIDI can be a bit of a chore. Wading through byte after byte of MIDI data and translating messages into something your application can work with is hardly the most exciting task in your programming life.

Why not have something do the job for you?

MAL can handle the task of mapping MIDI controls – and even keyboard notes – to whatever your application requires.

This is achieved using OSC (Open Sound Control), which also means that the MIDI devices need not even be attached to the computer your application is running on. It also means that you don't have to have one application to perform the mappings and receive the messages (you could write a mapping client that does nothing but map controls, for example.)

One of the really neat features of MAL is the ability to learn mappings. Place a MAL port into learn mode with a designated target, and a control mapping can be achieved simply by the user manipulating the control they wish to map.

### How to Build

You will need the latest JACK sources (preferably from SVN), along with liblo (an OSC library.) These need to be installed in a system-wide location as MAL needs to be able to include them without specifying the full path.

There is no fancy build system. Simply type **make** and hit the enter key. This will build **mal-server**, the server application.

### Comments and Contributions

Want to chat about MAL? You can get in touch with me via my website at [www.silverblade.co.uk](http://www.silverblade.co.uk) and I can also be found frequently on the **#lad** IRC channel on **FreeNode**.

I welcome suggestions, constructive criticism and bug reports.

### Planned Features / To Do / Known Limitations

There is currently no way to save mappings or enumerate existing mappings for a port.

It may be a nice touch in future to allow the uploading of a custom configuration for MIDI surfaces and allow each control to be referenced by a label (as opposed to MIDI control IDs.) At present, this is outside of MAL's intended feature set though.

System messages (0xF0 ~ 0xFF) are ignored (or at least should be!) Should there be a way to map System Exclusive messages etc?

## Data Formats

When a client requests a control mapping, it can specify one of the following formats (as the Format string parameter):

bool	The target will receive an OSC “T” or “F” (true/false, respectively)
float	The target will receive an OSC float with the value ranging from 0.0 to 1.0
char	The target will receive an OSC char with the value ranging from 0 to 127

Formats are specified as a string. If an invalid format is supplied, this is treated as an error and the associated request is aborted.

## Port Management

Each port represents a JACK port present on the MAL JACK client, and is essentially an individual MIDI bus. Applications that manage mappings typically won't add or remove ports – this feature is intended for use by a separate port/bus manager utility.

The intended purpose of MAL ports is to provide a way to differentiate between different MIDI controller hardware attached to the system. This means if two controller devices send the same control data, no conflict will arise. Of course, it's also feasible that multiple controller hardware devices will output different data to one another, thus they can safely transfer data to a single MAL port.

A client may request a list of the currently available MAL ports with this request. It must specify a URL and path for an OSC handler which takes a single string parameter (the port name.)

### **/mal/enumerate\_ports**

Return URL (string)

Return Path (string)

### **/mal/add\_port**

Port name (string)

### **/mal/remove\_port**

Port name (string)

## Control Mapping

Some controls are specified by method alone (eg: `aftertouch`, `pitch bend`, etc.) whilst others require a control ID as well. The actual control method is determined by the kind of message. For example, a `/mal/map_aftertouch` message will perform a mapping to all `aftertouch` messages whose channel and port match.

All mappings are associated with a port. This is a previously registered port via a call to `/mal/add_port` or one received in response to a `/mal/enumerate_ports` request. Mappings are also channel-specific, meaning that a control mapping on one channel will not be in effect for another channel.

As previously mentioned, some controls also require a control ID of some form.

In addition to this information, MAL needs to know what to map the control to! The target is made up of an OSC URL (eg: `osc.udp://localhost:16969/`) and a path (eg: `/foo/bar`) – this determines where MAL sends control change notifications, and is not interpreted by MAL.

Finally, each mapping requires a data format. This is the format of the sole parameter expected by the OSC handler. Data formats were covered earlier on in this document.

**/mal/map\_controller**

**/mal/unmap\_controller**

Port name (string)

Channel (char)

MIDI controller number (char)

Target URL (string)

Target path (string)

Data format (string)

*is this really necessary for unmapping???*

**/mal/map\_program\_change**

**/mal/map\_aftertouch**

**/mal/map\_pitchbend**

**/mal/unmap\_program\_change**

**/mal/unmap\_aftertouch**

**/mal/unmap\_pitchbend**

Port name (string)

Channel (char)

Target URL (string)

Target path (string)

Data format (string)

**/mal/map\_note**

**/mal/unmap\_note**

Port name (string)

Channel (char)

Note number (char)

Target URL (string)

Target path (string)

Data format (string)

## Keyboard Mapping

The entire keyboard can be mapped/unmapped with a specific target. When a note is played, the target URL will receive a message as specified by the target path, with a note number (char) and velocity (float) as its parameters.

### **/mal/map\_keyboard**

- Port name (string)
- Channel (char)
- Target URL (string)
- Target path (string)

### **/mal/unmap\_keyboard**

- Port name (string)
- Channel (char)
- Target URL (string)
- Target path (string)

## Control Mapping Learn Mode

When a port is placed into learn mode, all incoming data for that port is monitored. All existing mappings are ignored for the duration of the learning (this behaviour may change, or be configurable, in a future release – possibly via a couple of MAL OSC methods.)

The learned data is committed as a mapping when learn mode is stopped. If learn mode is instead cancelled, the learned data is discarded.

Only the most recent MIDI control data is retained (ie, the last received MIDI control data becomes the mapped control data.)

### **/mal/start\_learning**

- Port name (string)
- Target URL (string)
- Target path (string)
- Data format (string)

### **/mal/stop\_learning**

- Port name (string)

### **/mal/cancel\_learning**

- Port name (string)