to the beginning of my tracks in Live, it now has the index 0. So if I play with the same ID I stored for instance, I won't play with the track I just moved, but with the new track that is in the position of track 3 in the Liveset, which now has the index 2 (it previously had the index 1, but has now moved up one as a result of my track rearranging)

So, be careful with which output you take the ID from.

## 2.6 One shot request with *get* and live.object

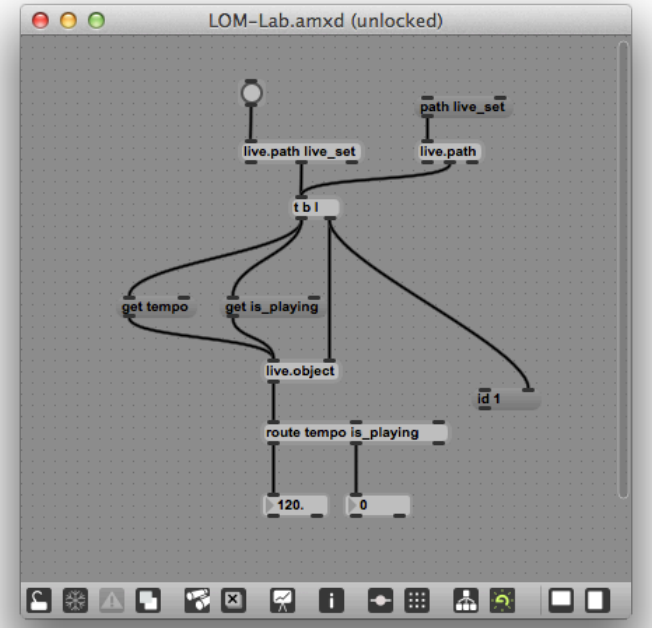
I like to experiment fast in Live using a basic setup with Max for Live. This one allows me to rapidly prototype things and to have it all in the same folder. Let's do that first.

### Creating a Max for Live rapid testing setup

- Open Live, create an empty liveset and save it with a name related to your test.
- drag'n' drop a Max for Live device from the MIDI Effect type in a track.
- Click on the edit button of this device
- Save this device in the same folder as your Live project

This is a basic guideline that you can follow to quickly create a **rapid prototyping environment**.

Now, edit your device and create a patch as in the figure below:




**LOM's testing**

This structure provides a simple way to test and understand IDs, paths, and to make one-shot requests to Live by using **live.object**.

There are two ways of using **[live.path]**:

- setting the path as an argument. Arguments are assigned to objects by putting a space after their name, then the arguments itself.
- sending a message containing the keyword *path* followed by the path itself.

Both are syntactically equivalent, but the second is more interesting when we require **dynamic systems**. For instance, if we want the patch to be able to make requests about multiple tracks, we will need to build paths dynamically depending on the track we need information from. We cannot do that when the index of the track is set as an argument.

Here, from our small Max for Live device, we want Live to tell us what the tempo is and if the transport is playing or not.

If we check the LOM documentation, we can see that tempo and playing status are available as properties of the object Song. Indeed, these values depend on the liveset itself.

I made a small cut to show specifically the data we need to see within one table:

**Song**

This class represents a Live set. The current live set is reachable by the root path **live_set** .

**Canonical Path**

live_set

**Children**

| *Name* | *Type* | *Access* | *Description* |
|---|---|---|---|
| appointed_device | Device | get, observe | The appointed device is the one used by a control surface unless the control surface itself chooses which device to use. It is marked by a blue hand. |
| cue_points | list of CuePoint | get, observe | Cue points are the markers in the arranger to which you can jump. |
| master_track | Track | get | |
| return_tracks | list of Track | get, observe | |
| scenes | list of Scene | get, observe | |
| tracks | list of Track | get, observe | |
| view | Song.View | get | |
| visible_tracks | list of Track | get, observe | A track is visible if it is not a subtrack folded in. Hiding tracks be scrolling them out of view is something completely else. |

**Properties**

| | | | |
|---|---|---|---|
| is_playing | bool | get, set, observe | 1 = the Live transport is running. Can be used to stop or start the Live transport. |
| tempo | float | get, set, observe | Current tempo of the Live set in bpm, 20.0 ... 999.0. The tempo may be automated, so it can change depending on the current song time. |

**We need the object Song here**

All that we need are these two properties :

- tempo
- is_playing

**tempo** data type is *float*, which basically means it is a number with a decimal point i.e 120.50. We see that we can also perform a get, a set and even an observe request on this property.

**is_playing** data type is a *bool* that stands for a boolean. Its value can be *true* or *false*. We can also perform a get, a set and an observe request on this property.

Now, we know which properties we can request and to which object we need to perform those requests.

Let's check the canonical path of the object Song: **live_set**

Let's pass it to **[live.path]** and this one will produce a result formatted like this: **ID <number>** where <number> is the numerical ID of the requested property.

## BE CAREFUL OF ID 0

If ID 0 is the result of **[live.path]**

Then it means that no object fits with the path passed to the **[live.path]**

ID 0 is often a clue saying you made an error in the path.

Object **[trigger]**, abbreviated as **[t]**, provides a way to control the outgoing flow of messages. Here, **[t b l]** means that if the trigger object receives *something*, it will process it as a list that will be transmitted from the rightmost outlet then after this operation, it will send out a bang from its leftmost outlet.

The message element linked to its rightmost outlet is only here for control and display purposes, it shows us what ID is sent out of **[live.path]**. This value is transmitted to the **[live.object]** which will be *linked* to the object's instance related to the transmitted ID, here: the Song object.

From this moment, we can transmit multiple requests to **[live.object]** and it will process them accordingly to the LOM, relatively to the object Song, that means the liveset itself.

Then we send two messages to it:

- **get tempo**
- **get is_playing**

And **[live.object]** answers with the two responses :

- tempo 120.00
- is_playing 0

**[route]** object provides a way to route incoming messages to many outlets depending on a prefix (tag) in the received messages. It also un-tags these incoming messages (here it removes the *tempo* and *is_playing* prefixes)

Play a bit with this patch.

Launch some requests, observe the results, then change the tempo and activate the transport in Live.

It works fine but we need to relaunch the requests to get the updated values.

### Requesting values occasionally

1/ find the property you want to know the value of

2/ find the object providing this property and remember the object's canonical path (the canonical path can be found under the object name in the LOM documentation)

3/ give this canonical path to live.object

4/ request the live.object with messages like : **get <property name>** and this one will answer with messages like **<property name> <current value of the property>**

This is good! But we have to make a request every time we change the value and if we change the value we know the new one anyway, so why would we have to request it… ?

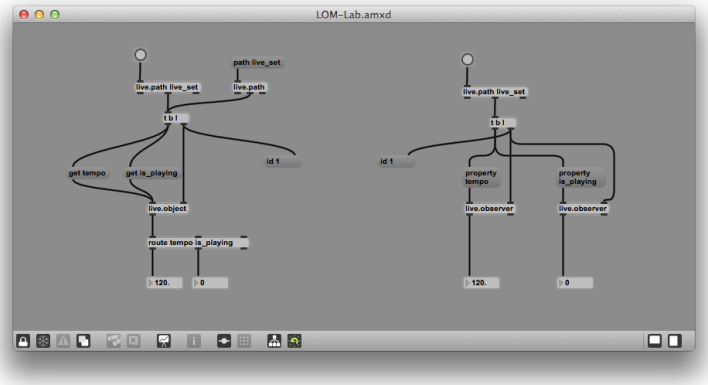
It doesn't make sense because *what we need is the updated tempo value at anytime.*

## 2.7 Observing properties in Live with live.observer

As mentioned previously, **[live.observer]** provides a way to be informed about a property's value change whilst the property is observed by the **[live.observer]** itself.

This means, we can give an ID to **[live.observer]**, we tell it what property we need it to follow, and it does the job.

Here is the corresponding example patch:




**Observe tempo & transport's state**

I have consciously left the previous patch to the left with the **get / [live.object]** technique.

We can see that the ID itself is the same in both cases. This is totally normal (and safe) as we can provide the same path which navigates to the Song object to both **[live.path]** objects.

Here, we have the **[t b l]** object again, and it transmits the ID to the **[live.observer]**.

Please, be aware of this:

**[live.observer]** can ONLY observe one property at a time !

If we need to observe two properties, we need two **[live.observer]** objects !

Then, to define which property the **[live.observer]** should follow, we send it this message:

**property <property name>**

From this moment, **[live.observer]** transmits from its outlet the value of the property. If this latter changes, a new value is sent out.

Play with it as you did with the previous patch.

Change the tempo, start & stop the transport while looking at the patch. You will see the **[live.observer]** objects update with the same correct values.

### Observing a property

1/ find the property you want to know the value of

2/ find the object providing this property and remember the object's canonical path

3/ pass the canonical path to **[live.path]** which finds the ID for **[live.observer]**

4/ send the message **property <property name>** to the **[live.observer]** which will send out the current value of the property.

This is the way to always be informed with up to date properties' values.

## 2.8 Calling objects' functions with live.object

We are going to learn now how to call functions.

When we read LOM's documentation, we can see that each object owns a set of functions.

We are going to create a small patch that provides a way to trigger clips.

### Seeking the object required

Let's scroll through the LOM documentation. We can see an interesting object, the **ClipSlot**.

Indeed, without worrying about whether there is a clip or not inside the clip slot, we can use the function **fire** and this function will, as read in the LOM, trigger the clip if there is one, or trigger the stop button of the clipslot if there is not.

This is exactly what we want to.

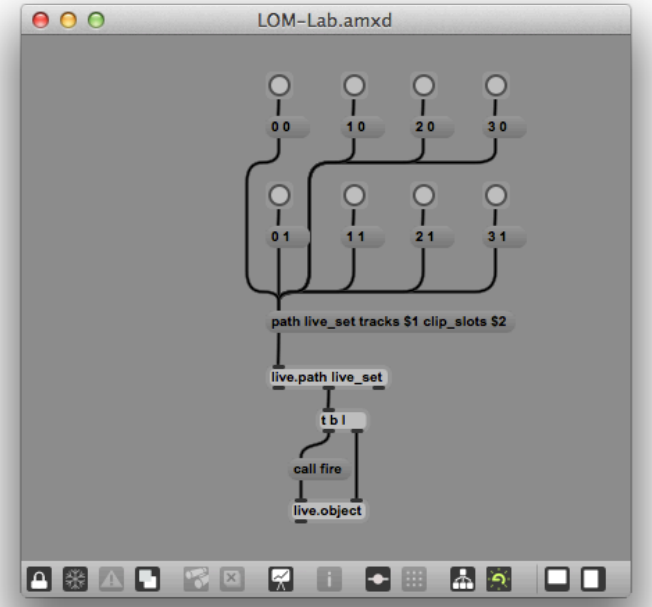
The canonical path to reach each clip slot in our clips' grid is:

**live_set tracks N clip_slots M**

with N being a track's index & M as scene's index (N and M are the coordinates of the clip slot, starting from zero in each dimension)

### Building the patch

Here is a possible example patch.




**Triggering clip slots**

Here the canonical path has been built dynamically.

This means that when we push each button, the static message placed under the buttons will send out the following message:

**path live_set tracks $1 clip_slots $2**

$1 & $2 are what we use to call **placeholders**. These are variables (known as replaceable arguments) that get their values from incoming messages, in the same order of the incoming list.

I chose to put the index of the track first, then, the index of the scene (our clip slots coordinates)

For instance, if I send the message (2 1) to my message path, this will transmit **path live_set tracks 2 clip_slots 1** to **[live.path]**

And **[t b l]**, after having transmitted the ID received from the **[live.path]**, will trigger a bang sent to the message (call fire) and then, as soon as this message goes into the **[live.object]**, the function fire will be called and the corresponding action triggered.

Here, the clip slot is triggered *exactly as* if we were to push its stop button or if there was a clip the play/stop button of the clip.

Put some clips in the liveset between scene 0 and 1 and track 0 to 3 and play with the patch.

### Calling objects' functions

1/ find the object and its function you want to call

2/ pass the canonical path of the object to **[live.path]** which finds the ID for **[live.object]**

3/ send the message **call <function name>** to **[live.object]**

Functions act on the liveset and thus modify, as a consequence, other properties.

For instance, triggering a clip with the transport stopped will also start the transport. The value of the is_playing property of the object Song will change to 1.

But we can also change some properties' values without using functions.

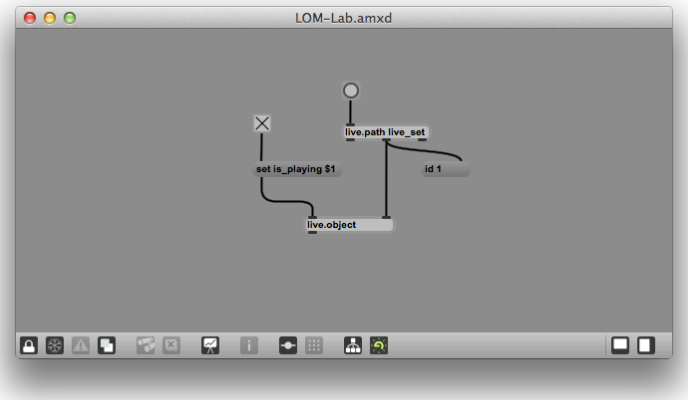
## 2.9 Changing properties' values by using *set* & live.object

We are going to create a button to start and stop the transport by changing the value of the property **is_playing**.

Don't forget to check what you can and cannot do with each property you want to use!

Indeed, **is_playing** allows us to **get**, **set** and **observe** and this is also why I'm often using it during my training sessions in order to illustrate how we can use all of a property's features, globally.

Look at that patch:




**Apply a set on a Live object**

It looks strangely like the first example we built with **get**.

Here, in place of **get is_playing**, we have:

**set is_playing $1**

This is a basic way to change a property's value.

**[toggle]** provides an easy way to change a value from 0 to 1 and from 1 to 0 with a nice and cheap user interface: a check button.

Push it several times, it will trigger Live's transport.

### Changing a property's value

1/ find the property you want to modify

2/ find the related object and its canonical path

3/ send this canonical path to **[live.path]** which will find the ID for **live.object**

4/ send the message **set <property name> <new value>** to **[live.object]** and the property's value will be modified.

We can do this for continuous parameters too, I mean, a parameter that takes a range of values and not only on and off, or 0 and 1.

## 2.10 Controlling continuous parameters in Live with live.remote~

Imagine that you need to control the value of a track's mixer panning of track 3 and you want to make it follow a continuous sine wave, for instance.

We can totally do that with **[live.object]**.

But I must warn you about this, and would not encourage you to do it!

*Why ?*

Because when we do this with the **set / [live.object]** technique, all modifications are recorded in Live's UNDO history

By using **set / [live.object]** technique with continuous parameters, we would pollute very quickly the UNDO history.