Fixture Libraries for Visualisation

Lars Wernlund
Vangelis Manolis

AtlaBase
www.atlabase.com

November, 2020

Part I
Introduction

In the past year we have seen an increase in the interest for open fixture library formats. Several initiatives exist, such as the Open Fixture Library, GDTF and the ESTA Next Gen Luminaire Library study group.

Meanwhile we at AtlaBase have been working on creating fixture libraries for both visualisers and consoles for close to 15 years. While we welcome the industry's newfound interest in open libraries we can see that the initiatives have so far been quite console oriented. As we recognize that few will have any hands-on experience with the requirements of a library capable of catering visualisers we would like to contribute with some insight into how we work and what we would consider as key features of such a library.

Some might assume that we would not be interested in contributing to open library formats since AtlaBase is a supplier of such library data. However, the fact is that we don't believe that an open fixture library format by itself is a replacement for a library supplier and our library editors would prefer gathering data from structured file formats over ambiguous textual data in PDF files. Furthermore, many fixture manufacturers will require assistance in using whatever new file formats are adopted by the industry, which by itself presents a business opportunity for us.

In this white paper we outline the data model that we use for fixtures, both for their general properties and the representation of their DMX modes. In the conclusion we summarise the two factors we would consider most critical in fixture libraries for visualisers.

This white paper is technical and the reader is assumed to have at least a basic understanding of programming concepts and data structure.

Part II
Data model

Let us begin with a brief overview of the kind of fixture information that AtlaBase gathers, starting with general properties defining a fixture's characteristics before we dive into DMX control.

In reality AtlaBase gathers library data about more than just fixtures, including filters, gobos, trusses and plot symbols, but we will only be looking at fixtures in this paper.

1 General properties

The following are the most important groups of fixture properties.

1.1 Parts

We represent the physical model of a fixture as a hierarchy of parts. Parts are connected by axes, which may or may not be motorized. Each axis has an angular range, direction and some speed constraints.

Each part has a 3D model that represents its geometry. At the very least this is a set of polygons consisting of vertex coordinates and normals. Normals are important in order to avoid fireflies / flares that could occur from flat surfaces catching light as the fixture rotates or the viewer is moving. Ideally the 3D model would also have UV coordinates, tangents and materials following PBR (Physical Based Rendering) standards. Expectations of the appearance of fixtures in visualisers are high and it is often necessary to model fixtures from scratch for this purpose rather than attempting to convert or simplify manufacturing drawings.

Finally, it is also vital to have a well defined set of rules for the orientation of parts in relation to a fixtures home location. Additional information such as the location of a fixtures on-board menu, display and pigtail may be required.

1.2 Light sources

Any part of a fixture may have one or more light sources attached to it. Each light source models a light emitting technology and requires a description of its luminous output. The terms used may vary but are typically field angles, generic intensity distribution curves, lumens and peak candela. Ideally intensity distribution data can be drawn from IES or EULUMDAT files, especially for architectural fixtures.

Some light sources contain multiple apertures, sub instances on the plane of the light source. A few examples where this is commonly found are molefays, strip lights, LED washes and LED strobes. Each aperture has an x/y location on the light source's plane and a cell number which plays an important role in DMX control.

1.3 Light source colours

It is important to have the colour characteristics of each light source well defined, either as a white colour temperature, a CIE colour or a full spectral distribution. For light sources with multiple emitters, the relative power between the emitter colours is also important as this may affect colour mixing.

If a fixture is equipped with virtual colour mixing models such as RGB or HSI, it is important to have the primaries of the virtual RGB colour space defined as well as the target white point.

1.4 Wheels

Many fixtures are equipped with wheels that can be fitted with gels and gobos. Sometimes the entire wheel itself is a gobo, as in the case of animation wheels.

Colour information for filters is required. This could be a CIE colour and the characteristics of the light source used to measure it, typically a standard illuminant or a correlated colour temperature.

Image of gobos and animation discs are required. In the case of animation discs it is also relevant to know the physical size of the animation disc in relation to the diameter of the light path in the fixture, so that a correct amount of the pattern can be applied to the beam at any given point.

2 DMX modes

The fact that AtlaBase has been supplying Capture, a visualiser, with libraries since day one has had a profound effect on how we work with DMX modes.

In order for a visualiser to maintain a simulation of a fixture it needs to have a well defined model of all its possible components and their behaviour. You could think of it as a super fixture that has all known features on the market. In reality most fixtures will only use a fraction of them, but the full model needs to be in place as a feature not included could not be simulated.

As a result of this we are in a constant race to model all new emerging technologies. All new concepts must be analysed and understood on a detailed level in order for us to be able to create a theoretical model of it that could be applied to fixtures from multiple manufacturers.

Unfortunately there are plenty of cases where this fails and we have to revert to describing the effects of a DMX range using English text. This is necessary for us in order to be able to supply lighting consoles with libraries, but ultimately it does represent a failure for the library of a visualiser. We use it as a failsafe and work systematically on revisiting such cases in order to find the correct solution.

2.1 The command model

To model the DMX modes of a fixture we rely on something we call commands. Each channel consists of a listing of DMX levels and ranges that trigger various commands. Each command is essentially an instruction that changes or affects the behaviour of a fixture in some way.

It is important to understand that we do not divide a channel into non-overlapping DMX level ranges and assign each range a unique kind of behaviour. Commands may be triggered by overlapping DMX levels and ranges.

2.2 Example – iris

A common fixture feature is the iris, a mechanically variable aperture often found in fixtures capable of pattern projection. It can be set at a specific position as well as to be strobing. When strobing, the pattern it strobes in, the frequency at which it is strobing and the opening and closing transition type can be set.

Our iris commands are:

```
enum iris_mode { linear | strobing }
enum strobe_pattern { periodic | random |.. }
enum strobe_transition { ramp_ramp | snap_ramp | ramp_snap |
snap_snap }

set_iris_mode(iris_mode mode)
set_iris_linear_position(float percent)
set_iris_strobe_pattern(strobe_pattern pattern)
set_iris_strobe_frequency(float frequency)
set_iris_strobe_transition(strobe_transition transition)
```

Usually these commands are only found within a single channel, but sometimes they are spread out over multiple channels.

Here is what the Robe Robin MMX Spot's Iris channel in Mode 1 looks like:

```
@0-191: set_iris_mode(linear)
@192-255: set_iris_mode(strobing)
@0: set_iris_linear_position(0%)
@1-179: set_iris_linear_position(0% - 100%)
@180-191: set_iris_linear_position(100%)
@192-247: set_iris_strobe_pattern(period)
@248-255: set_iris_strobe_pattern(random)
@192-219: set_iris_strobe_transition(snap_ramp)
@192-219: set_iris_strobe_frequency(0% - 100%)
@220-247: set_iris_strobe_transition(ramp_snap)
@220-247: set_iris_strobe_frequency(100% - 0%)
@248-249: set_iris_strobe_transition(ramp_snap)
@248-249: set_iris_strobe_frequency(100%)
@250-251: set_iris_strobe_transition(ramp_snap)
@250-251: set_iris_strobe_frequency(0%)
@252-253: set_iris_strobe_transition(snap_ramp)
@252-253: set_iris_strobe_frequency(100%)
@254-255: set_iris_strobe_transition(snap_ramp)
@254-255: set_iris_strobe_frequency(0%)
```

For reference, here is the table of the Iris channel from Robe's manual:

	Iris	
0	Open	step
1 - 179	From max.diameter to min.diameter	proportional
180 - 191	Closed	step
	Pulse effects with Iris blackout:	
192 -219	Pulse opening from slow to fast	proportional
220 - 247	Pulse closing from fast to slow	proportional
248 - 249	Random pulse opening (fast)	step
250 - 251	Random pulse opening (slow)	step
252 - 253	Random pulse closing (fast)	step
254 - 255	Random pulse closing (slow)	step

2.3 Benefits of the command model

In the previous iris example, no DMX level range had to be described using the English language. A lighting console with a Russian or Japanese user interface could produce its own descriptions of all functions as well as have a complete understanding of each DMX level.

With our iris commands, a total of 16 possible combinations could be meaningful for any given DMX level, but there is no need to define these combinations in advance. Using commands requires no extra effort to model channels controlling multiple gobo wheels or preset channels that combine colour wheel positions with virtual RGB colours.

A fixed set of commands also lends itself well to an op-code based assembly-like approach. With a known amount of maximum arguments a denormalised can easily be constructed to hold the data for a DMX channel in a very compact binary form. Each command can be reduced to something like this:

```
command(int opcode, int param1, int param2, float param3,
      float param4, string text)
```

2.4 Drawbacks of the command model

With flexibility and power comes complexity. In order to reproduce the table from Robe's manual in the iris example a denormalising algorithm needs to be applied. While we do have such algorithms in store, writing them from scratch is not trivial, especially if one does not have any previous experience with data transformations. A glossary also needs to be established for all fixture features as one can no longer rely on the albeit convenient but inconsistent linguistic preferences of fixture manufacturers.

2.5 Further examples

2.5.1 Wheels

This previous iris example is relatively simple, but the principles of it apply to many other components of our fixture model as well.

The following are the commands related to the rotation of a wheel, that is not the gobos inside the wheel, but the wheel itself.

```
enum wheel_mode { indexed | continuous | spinning | .. }
enum wheel_direction { forward | reverse }

set_wheel_mode(int index, wheel_mode mode)
set_wheel_spinning_direction(int index, wheel_direction direction)
set_wheel_position(int index, wheel_mode in_mode, float position)
set_wheel_speed(int index, wheel_mode in_mode, float rpm)
set_wheel_shutter(int index, bool enabled)
```

The `in_mode` construction is of particular importance. The current state of each wheel must be tracked in order to know whether a `set_wheel_position` or `set_wheel_speed` command is applicable. This is how we approach modality and conditionality in DMX control.

2.5.2 Dimming and colour mixing

The following are the most common commands related to dimming and additive colour mixing. Two layers of mixing are provided and each command (except the master intensity) directly addresses a cell. Each cell maintains its own two colour mixers which can be in one of a number of modes.

```
enum cell_layer { individual | group }
enum cell_mode { Direct | CT | RGB | HSV | Preset .. }

set_master_intensity(float intensity)
set_cell_intensity(int[] indexes, cell_layer layer,
     float intensity)
set_cell_mode(int[] indexes, cell_layer layer, cell_mode mode)

set_cell_direct_white(int[] indexes, cell_layer layer,
     float intensity)
set_cell_direct_red(int[] indexes, cell_layer layer,
     float intensity)
set_cell_direct_green(int[] indexes, cell_layer layer,
     float intensity)
set_cell_direct_blue(int[] indexes, cell_layer layer,
     float intensity)
set_cell_direct_aux1..5(int[] indexes, cell_layer layer,
     float intensity)

set_cell_virtual_white(int[] indexes, cell_layer layer,
     cell_mode in_mode, float intensity)
set_cell_virtual_red(int[] indexes, cell_layer layer,
     cell_mode in_mode, float intensity)
set_cell_virtual_green(int[] indexes, cell_layer layer,
     cell_mode in_mode, float intensity)
set_cell_virtual_blue(int[] indexes, cell_layer layer,
     cell_mode in_mode, float intensity)
set_cell_virtual_white_ct(int[] indexes, cell_layer layer,
     cell_mode in_mode, float kelvin)
```

Just as wheels track their current mode, so do cells as they can switch between different colour mixing models. A common example of such switching happens in virtual colour wheel channels that override other colour mixing channels.

Part III
Conclusion

We believe it is necessary to break down and create a model of all possible fixture behaviours. By definition, this model will be very close to that of a visualiser's simulation engine.

1 Strictness

Any new emerging technologies have to be modelled with great haste. Failure to do so will result in creative (ab)use of the existing model. Any provisions to allow fixture manufacturers to define custom behaviours should be kept to an absolute minimum.

2 Normalisation

Any DMX level must be able to trigger a combination of behaviours. Attempting to catalogue such combinations in advance is futile and unrealistic as we know there are thousands of such combinations. This may be inconvenient for console libraries, but it is necessary for visualiser libraries.