DataspaceLib – Functional Specification

Timothy Place

2009-03-29

Abstract

1 Overview

The **DataspaceLib** is a library within the Jamoma environment for performing conversions between data whose values may be represented in different forms or units. The DataspaceLib is not used directly by end users in Max, but rather is integrated into objects that need its services. For example, the jcom.dataspace object can be used to do simple conversions and the jcom.parameter object can access the DataspaceLib internally to allow parameter values that are specified in many different units.

This spec is far from complete.

2 Use Cases

2.1 Case 1: Prometheus and the Temperature of Fire

Prometheus, having stolen fire from the gods wants to measure how hot the fire is and then convert between various temperature units (Kelvin, Celsius, Fahrenheit) to teach them to human across various places of the world. He needs to convert between temperatures simply and directly just using jcom.dataspace.

2.2 Case 2: Echo the Nymph

When Hera curses Echo she needs to be able to set the amount of time for the echo. Hera wants to be able to control the delay time of the echo in different time units: seconds, milliseconds, samples, Hertz.

2.3 Case 3: Brita the Filterer

Brita needs to have a filter module. The parameters of the filter in the filter module are frequency, q, and gain. They need to be displayed and controlled using a filtergraph object and number boxes or sliders. The frequency and gain both must be specifiable in various units.

2.4 Case 4: Lenny the +10dB producer

Lenny, a successful producer from San Francisco, doesn't like midi numbers and feels more comfortable with dB (especially with +10 dB). Therefore he wants to change globally all gain UI elements in Jamoma (knobs, textslider, numberboxes) from default Jamoma-MIDI to dB.

2.5 Case 5: Jean Luc and the final frontier

Jean Luc has a mission: space - the final frontier. He navigates sounds through a three dimensional cartesian room. Sometimes, especially after a bottle of wine from the Galliac region, canards from heaven appear and spinning spherically around his head. In these moments, he would like to describe his sonic experience in a similar way - with spherical coordinates.

3 Data Flow

3.1 DataspaceLib

The DataspaceLib itself is already designed and implemented, so there is not a compelling need to create a blueprint for it here.

3.2 Integration with jcom.parameter

The jcom.parameter (which includes jcom.message for all intents and purposes) will have the following attributes:

3.2.1 @dataspace

Define the dataspace, which in turn determines what units are available.

This attribute should be readonly (see section 3.2.3), but we don't have a way right now to control it.

3.2.2 @unit/active

This is the primary input and output unit for the module/parameter.

Default value is the neutral unit of the dataspace.

active unit is applied to presets which are saved as well (and the unit is always written into the file)

implies that the cuelist and mappers will always use the active unit

Changing the active unit dynamically during runtime may have unintended side-effects. So it might be that we wish to consider this attribute readonly...

3.2.3 @unit/native

This is the internal unit used to send values to the algorithm.

Default value is the value of @unit/active.

This attribute is readonly – meaning that it is set by the author of a module at the time that the module is created, and cannot be changed on-the-fly at runtime.

3.2.4 @unit/display

This is the unit used to communicate with the UI of the module (number, slider, etc).

Default value is the value of @unit/active.

This attribute might be changed through an additional inlet/outlet to jcom.parameter?

3.2.5 @unit/override

This is a very special unit, which is used when there is input from the user via a message, and that message contains a unit which is not the active unit. In this case the specified unit of the message (the override unit) is used temporarily to drive the parameter.

Default value is the value of @unit/active.

This attribute is completely hidden from the user and is not directly accessible in any way.

example message: [gain -6 db]

The way the delay object works in this example is an example of the override unit. In this case the native/active units are samples, but the override is ms.

4 Open Issues

4.1 Integration with jcom.parameter

4.1.1 Inlets and Outlets

The problem is that there are currently 2 uses for that second outlet of which I'm aware:4:33 PM

1. it controls something in the algorithm directly (which suggests native unit)

2. it controls some part of the display for which the possibly deferred or otherwise treated first outlet is insufficient (to refresh or calculate filtergraph coefficients or something). This suggests it should stay in the display unit.

I am wondering now if we just need a whole bunch of outlets, one for each type or something? (Nils does not like this, and Pascal either) Pascal thinks that having the first outlet in the display unit, and the second one in the native unit sounds good... Could anyone please provide an example where this problematic ?

4.2 The conversion to native unit: jcom.parameter or jcom.in?

Nils Asks the following:

The native unit, which goes to the algorithm.... this only appears at the jcom.in ?

I think towards a MVC it makes sense to dedicate the active unit to the jalg - means that it only appears at the output of jcom.in

Tim Responds:

In general, I don't think the MVC thing will cause us any grief because the algorithm and the parameters are all together in the model. The one that is trickier is the display because there could be multiple views – but I think we should play östereich and stick our heads in the sand on that for now.

Interesting thought about where the actual code goes though. In the current architecture it does make sense that the active to native conversion would be located in the jcom.in, which I hadn't considered before.

I think it will make more sense in the future (MVC) to keep it in the parameter though.

Pascal asks : will there be a jcom.in at all in the MVC future, if the parameter is in the model ?

And I think that this discussion about the MVC might clarify your previous question, and that yes the 2nd outlet of jcom.parameter should be spitting out the native unit

The problem is that there are currently 2 uses for that second outlet of which I'm aware:

1. it controls something in the algorithm directly (which suggests native unit)

2. it controls some part of the display for which the possibly deferred or otherwise treated first outlet is insufficient (to refresh of calculate filtergraph coefficients or something). This suggests it should stay in the display unit.

4.3 Mappers

Nils:

I can see that one would like to work for mappings with units independently from whatever unit is currently active.....but we can work on that later if necessary ?

Tim:

We had a good discussion about this in Albi (but it isn't written down anywhere, naturally). If a module wishes to work in a different unit, then it is incumbent upon that module to do its own work. So a mapping module can query a parameter to find out what dataspace it has, and what the active unit is. Then it can use a jcom.dataspace object to do whatever mapping is requested by the user.

Nils:

that seems easy to implement without breaking too much. However, you may end up with a chain of unit conversions such as from unitA to unitB to unitC to unitA that only causes rounding errors and processing....but that's probably only a theoretical issue.

Tim:

I have been more concerned about the efficiency than about rounding errors, but theoretically both are possible. Fortunately there will be no conversions unless you ask for them.

It may be possible in the future to optimize as well. For example, if the graph can be detected as containing unitA to unitB to unitC to unitA then it could optimize itself to just convert directly between the bounding units. As with any such optimization, I would place it as something to be done in the distant future – after we have a fully functioning and field tested system in place.

4.4 UI elements

for certain UI-elements such as jcom.textslider or dial, a range has to be defined. This range needs to change when we change between different units.
One idea would be to implement Jamoma-specific UI-elements (or just wrapper for existing UI elements) with consistent attributes in order to modify them directly from a jcom.parameter/message

4.5 range/bounds

To what unit (active/native/display/override) relates the range/bounds attribute ?

4.6 More Use Cases

Another use case (or set of them) is what happens when reading in presets with no unit defined. I'm not sure if there are good examples or not. Probably there are in the spatialization modules somewhere.