

FrameLib: Alpha 0.2 Documentation

Alex Harker

Generated on 18/9/2017 at 19:33

Contents

1	Introduction	3
1.1	Preliminaries	3
1.2	What is FrameLib?	3
1.3	How is this an ‘Alpha’ Version?	3
1.4	How Can I Help?	3
2	Key Concepts	4
2.1	Schedulers and Timing	4
2.2	Parameters	4
2.3	Types of Frames	4
2.4	Multi-channel Expansion	5
3	Getting Started	6
3.1	List of Current Schedulers	6
3.2	Connecting Inputs and Outputs	6
3.3	Getting Audio In and Out	6
3.4	Setting Parameters	6
3.5	Setting Fixed Inputs	6
3.6	Control from Max	7
3.7	Getting Object Help	7
4	Learning More	8
4.1	Demo Patches	8
4.2	The Objects	8
5	Current Object List	9
5.1	Schedulers	9
5.2	Unary	9
5.3	Binary	9
5.4	Vector	9
5.5	Generators	9
5.6	IO	9
5.7	Mapping	9
5.8	Max Specific	10
5.9	Parameters	10
5.10	Routing	10
5.11	Spectral	10
5.12	Storage	10
5.13	Timing	10
5.14	Filters	10
5.15	Spatial	10
6	Alpha Aspects of FrameLib	11
7	Changes in Alpha 0.2	12
8	Changes in Alpha 0.3	12

1 Introduction

1.1 Preliminaries

This document is designed to go with alpha releases of the Max version of FrameLib (which is currently the only extant version). The documentation is currently intended to give just enough information to get experienced Max users started and explain basic concepts, and as such is far from fully comprehensive. To install simply place the FrameLib folder in the your Packages folder. This isn't a full package install (just externals), but there is no need to create the inner folder structure manually.

1.2 What is FrameLib?

FrameLib is a DSP system designed to allow quick modular constructions of frame-based networks. These networks can resolve time at a highly accurate subsample level and can run at different rates, and with different frame sizes across a single network. This allows mixing of different representations (e.g. time and frequency-based representations) in a single network, efficient processing when data sizes are varying and efficient, logical expression of tightly-timed DSP constructs. The underlying C++ framework can theoretically be hosted in any block-based DSP environment, but the current user-facing version targets only Max as a set of externals. Most of you should already basically know what it is (otherwise you wouldn't be testing), so that will do for now...

1.3 How is this an 'Alpha' Version?

In general the software should be relatively stable, but there are several items that are subject to change in terms of usage, which means **you should not use FrameLib for any large/final version projects yet!** You would do so at your own risk. There is a section later in this documentation that covers items subject to change and specific questions about these items.

1.4 How Can I Help?

Thanks for asking! Please comment on the usability of objects, features, points of confusion. Object ideas are also welcome, but are lower priority. Comments on the open questions in the alpha section are particularly welcome. Offers of help with documentation (or help patches) would be good, although most likely any help patches are a little way off yet. Obviously, let me know of any bugs you might find (crash logs and repeatable steps to reproduce, please). Note that the testing builds have some asserts left in, meaning that FrameLib will crash if it detects an unexpected issue - hopefully you won't have this happen, but this lets me test my assumptions 'in the wild'. Developer documentation (for those wishing to write C++ objects) will come later, via doxygen, when I get to adding the relevant comment hooks (and figure out how to get doxygen to document only the things I want in the right way).

2 Key Concepts

2.1 Schedulers and Timing

The most important concepts to understand in FrameLib are:

1. **Everything** is strongly timed (This timing is accurate to 2^{-64} of a sample).
2. **Nothing** happens without a timing source.
3. Objects that act as timing sources are called *schedulers*.

Schedulers trigger frames which then are then further processed by other objects. All FrameLib objects can accept inputs that are running at different rates (regular or irregular) and will resolve the timing between inputs correctly. Frames arriving at the same time are processed once only together. When frames arrive at different times they are processed against the latest arrivals at other inputs.

In order for your patch to do anything at all it will need at least one scheduler. You can also connect schedulers in any combination in a network and time should be resolved correctly.

Note that not all inputs to a FrameLib object will necessarily trigger processing (create output). This can vary between objects, and also over time (although this is used sparingly for specific functionality, such as the *fl.select~* object). This is seen most commonly on parameter update inputs, which are used to update object parameters....

2.2 Parameters

FrameLib objects have variables with state known as parameters. Parameters are referred to by name and are used for values that may not vary over a single frame (in this way they are unlike inputs, which are used when a value can vary over a frame). However, they may take more than one value (an array), or non-numeric values (for instance a string or an enumerator, which can be set by the name of the item, or its index number). The parameter system is used to change object behaviour, without requiring an unmanageable number of inlets, and to separate 'mode' type control from normal inputs. It is somewhat akin to the Max attribute system, but it operates in the strongly-timed way that FrameLib frames do.

Parameters may be set at instantiation time, or at frame rate using tagged frames (details of how this works in Max are given in section 3.4. Tagged frames are also described in more detail in 2.3). Importantly, for simultaneously arriving frames where one sets parameters and others trigger processing the parameter update is performed before the processing, so updates can be performed synchronously. Some parameters are available at instantiation (usually those that set the number of inputs/outputs or pre-allocate significant memory).

Each object maintains its own set of parameters which can be detailed within Max using the inbuilt reference system (see section 3.7).

2.3 Types of Frames

The inputs and outputs to FrameLib objects are 'frames'. Frames can have two basic types in FrameLib. The first is the most common and consists of a vector of numeric values (all doubles in underlying type). The length of this vector can take any length from zero upwards (so scalars are simply a single valued frame). The frame with no values (zero length) is used as a trigger message, and has useful function (similar to a bang message in max).

The second type of frame is a 'tagged' frame. This is used exclusively to transfer/set parameter values in FrameLib. A single frame can contain any number of tags (parameter names) followed by their values (of whatever type). This allows you to set multiple parameters in a single frame. Internally FrameLib has

functionality for concatenating tagged frames, so it is never necessary for an object to have more than one parameter update input.

2.4 Multi-channel Expansion

Technically, the objects hosted in Max objects carry not single frames, but multichannel connections each of which can consist of an individual stream of frames running at an independent rate and size. This is done to enable multi-resolution or multi-channel processing through a single network. If you wish you can ignore this functionality and treat each object like the inputs and outputs deal with a single stream of frames only. However, if you do wish to take advantage of this you can use the *fl.pack~* object to pack frame streams together, and *fl.unpack~* object to split them into single channel streams again. Other objects will resolve multichannel inputs by internally expanding to the maximum channel count. Where there is a mismatch between different inputs the lower channel count streams will be read modulo against the maximum count (so, for example, you can take a 16-channel input on the left of a *fl.times~* object and connect a single channel input to the right-hand side. The result will be a 16-channel output that consists of the 16 left-hand inputs all multiplied by the same right-hand stream).

3 Getting Started

3.1 List of Current Schedulers

Whilst this document is not an object reference, it is worthwhile giving more detail on the objects which act as schedulers.

<i>fl.interval~</i>	trigger regular frame - for irregular frames vary the interval over time.
<i>fl.audiotrigger~</i>	trigger frames using MSP non-zero values
<i>fl.perblock~</i>	trigger one frame per host audio block
<i>fl.future~</i>	schedule frames at specific points in the future
<i>fl.once~</i>	trigger one frame at the start of time, and never again

3.2 Connecting Inputs and Outputs

Only one object can be connected to a frame input. There is simply no one sensible way to combine two sets of frames in a generic manner. When FrameLib objects are connected directly this is enforced (you are not physically able to connect two objects). When you are using abstraction or subpatching this is not possible, but you will receive an error when dsp recompiles if you've connected two objects to a single input. One output can feed any number of inputs. Feedback loops are not allowed.

3.3 Getting Audio In and Out

The *fl.source~* object allows you to grab chunks of audio from MSP as frames. The audio you obtain suffers a latency equal to the size of the chunk you are grabbing. For output you will commonly want a *fl.sink~* object, which acts as an overlap-add buffer to the MSP signal chain. This suffers no latency. For single value outputs as control (for instance if you want to output the values of a frequency-domain analysis back in MSP) you will probably wish to use the *fl.trace~* object.

3.4 Setting Parameters

Parameters can be set in one of two ways, either at frame rate (if the object has a Parameter Update input), or at object instantiation. At object instantiation the syntax is currently:

either:

#parameter_name value(s) ...

or:

/parameter_name value(s) ...

within the object box. The *#* is used most commonly in the demos, but the slashes are also possible to see who prefers what. The *@* sign is not used to avoid confusion with max attributes (which show in the inspector and can be set in the max thread using messages, which FrameLib parameters cannot). In addition parameters can be assigned to a given argument number in which case they can be set from object arguments.

To set parameters at frame rate based on values in vectors, use the *fl.setparam~* object.

3.5 Setting Fixed Inputs

FrameLib objects in Max support input of fixed numeric values at object instantiation that are used for any subsequent calculations, rather than values updated at frame rate. This is accessed in one of two ways. Most binary operators (e.g. *fl.times~* or *fl.divide~*) allow you to directly type the value of either input in as one

or more arguments to the object (this mimics standard max object behaviour). However, as most objects use arguments to represent parameters (for ease), other objects require a special syntax for setting the value of a specific input. There are currently four possible notations to do this (with slashes or square brackets most likely to make the final cut):

either:

/input_number/ value(s) ...

or:

<input_number> value(s) ...

or:

[input_number] value(s) ...

or:

~input_number value(s) ...

Inputs are numbered from 1 (not zero!). You can enter vectors of any length.

3.6 Control from Max

As everything in max runs at frame rate you cannot just hook up max messages to control inputs. You must go through the *fl.frommax~* object. This object allows two modes of operation. In the first you can translate single numerical values or lists of numeric values into FrameLib vectors. In this case you simply pass max messages into the object. The last value(s) received will be sent as a vector when the object is triggered by a frame input.

In the second mode of operation a variable number of inputs allow for max messages to be translated into tagged frames. In this mode each input corresponds to a named parameter, and messages sent to each input will be stored until the next frame is triggered. At this point **all** stored parameter changes are sent to the output in a single concatenated frame, and cleared from the object's memory (unlike numeric vectors which will be repeated on subsequent trigger inputs).

3.7 Getting Object Help

There are a lot of objects and not time here to even list them. However, all objects support the **info** message. This currently prints reference type information to the max window. This is the only way to get help on individual objects, but should be compact but comprehensive.

Without any arguments message will print a description followed by info on inputs, outputs and parameters including ranges/enum types etc.). The operation/numbering of arguments is covered in the info text. You should note that the range and exact detail of parameters (although not the names or order) may change from one instance of an object to another. What you are receiving when you send the **info** message is technically info on the specific instance, and *not* the type of object.

You can also use any combination of the following to customise the info given:

- description (include the description)
- inputs (include info on the inputs)
- outputs (include info on the outputs)
- io (include info on inputs and outputs)
- parameters (include info on the objects parameters)
- quick (give brief versions of all info)

4 Learning More

4.1 Demo Patches

The demo patches supplied cover a set of simple and more complex behaviours. There is far more that you could do with FrameLib and these examples are biased towards spectral processing and granulation. These are the demos I've amassed for various presentations. They are not fully documented., but hopefully offer a solid starting point for how objects can be joined together to do something useful.

4.2 The Objects

A full object listing follows, grouped by function. Currently this is the best quick reference for what is included at present. For more info use the inbuilt help/info system.

5 Current Object List

5.1 Schedulers

<i>fl.audiotrigger~</i>	<i>fl.perblock~</i>	<i>fl.future~</i>	<i>fl.interval~</i>
<i>fl.once~</i>			

5.2 Unary

<i>fl.abs~</i>	<i>fl.acosh~</i>	<i>fl.acos~</i>	<i>fl.asinh~</i>
<i>fl.asin~</i>	<i>fl.atanh~</i>	<i>fl.atan~</i>	<i>fl.cbrt~</i>
<i>fl.ceil~</i>	<i>fl.cosh~</i>	<i>fl.cos~</i>	<i>fl.erfc~</i>
<i>fl.erf~</i>	<i>fl.exp2~</i>	<i>fl.exp~</i>	<i>fl.floor~</i>
<i>fl.log10~</i>	<i>fl.log2~</i>	<i>fl.log~</i>	<i>fl.round~</i>
<i>fl.sinh~</i>	<i>fl.sin~</i>	<i>fl.sqrt~</i>	<i>fl.tanh~</i>
<i>fl.tan~</i>	<i>fl.trunc~</i>		

5.3 Binary

<i>fl.atan2~</i>	<i>fl.copysign~</i>	<i>fl.diff~</i>	<i>fl.divide~</i>
<i>fl.equals~</i>	<i>fl.greaterthaneq~</i>	<i>fl.greaterthan~</i>	<i>fl.hypot~</i>
<i>fl.lessthaneq~</i>	<i>fl.lessthan~</i>	<i>fl.max~</i>	<i>fl.minus~</i>
<i>fl.min~</i>	<i>fl.notequals~</i>	<i>fl.plus~</i>	<i>fl.pow~</i>
<i>fl.times~</i>			

5.4 Vector

<i>fl.centroid~</i>	<i>fl.chop~</i>	<i>fl.crest~</i>	<i>fl.flatness~</i>
<i>fl.geometricmean~</i>	<i>fl.join~</i>	<i>fl.kurtosis~</i>	<i>fl.length~</i>
<i>fl.mean~</i>	<i>fl.medianfilter~</i>	<i>fl.pad~</i>	<i>fl.peaks~</i>
<i>fl.percentile~</i>	<i>fl.product~</i>	<i>fl.rms~</i>	<i>fl.shift~</i>
<i>fl.skewness~</i>	<i>fl.sort~</i>	<i>fl.split~</i>	<i>fl.spread~</i>
<i>fl.standarddeviation~</i>	<i>fl.subframe~</i>	<i>fl.sum~</i>	<i>fl.vmax~</i>
<i>fl.vmin~</i>			

5.5 Generators

<i>fl.count~</i>	<i>fl.random~</i>
------------------	-------------------

5.6 IO

<i>fl.sink~</i>	<i>fl.source~</i>	<i>fl.trace~</i>
-----------------	-------------------	------------------

5.7 Mapping

<i>fl.constant~</i>	<i>fl.convert~</i>	<i>fl.lookup~</i>	<i>fl.map~</i>
<i>fl.samplerate~</i>			

5.8 Max Specific

*fl.frommax~**fl.read~*

5.9 Parameters

fl.setparam~

5.10 Routing

*fl.pack~**fl.select~**fl.unpack~*

5.11 Spectral

*fl.convolve~**fl.fft~**fl.ifft~**fl.mutlitaper~**fl.window~*

5.12 Storage

*fl.recall~**fl.register~**fl.spatial~**fl.store~*

5.13 Timing

fl.now~

5.14 Filters

*fl.0dfsuf~**fl.onepolezero~**fl.onepole~**fl.resonant~**fl.sallenkey~*

5.15 Spatial

*fl.coordinatesystem~**fl.spatial~*

6 Alpha Aspects of FrameLib

- The object set is subject to change.
- All object features are subject to change (particularly which objects have parameter inputs).
- All object parameter names are subject to change.
 - here input on preferred formats (with or without underscores/capitals etc. would be helpful).
- Notation for parameters/inputs are subject to change.
 - here input on preferences is appreciated.
- The text displayed by the info object is subject to change.
 - here input on clarity/wording is appreciated.
 - input on what should be considered quick/comprehensive would be useful.

7 Changes in Alpha 0.2

- Fixed documentation for FFT object outlet/inlets.
- Fixed object info for objects using string replacement as a method for generating strings.
- Fix for objects with no ‘live’ inputs (no inputs that lead back to a scheduler).
- Removed unnecessary dspchain recompiles, related to patchline re-ordering.

8 Changes in Alpha 0.3

- Fixed testing for no live inputs to prevent crashes with audio-synced objects.
- [#12] Fixed binary templates to prevent overwrites in wrap mode with mismatched input lengths.
- [#1 and #14] Fixed *fl.medianfilter~* width clipping to prevent crashes.
- [#15] Fixed *fl.source~* to allow dynamic changing of the size.
- [#11] Updated *fl.fft~* and *fl.ifft~* to allow complex/full spectrum FFTs and to use the new HISSTools FFT code.
- Updated *fl.store~* now clears the buffer when audio is restarted.
- Refactored *fl.read~* to use template based interpolation (probably requires AVX) - also allow none as a user-selectable interpolation mode.
- Added interpolation and end behaviours to *fl.lookup~*.
- Corrected filtering on the multi-resolution demo.
- Fixed ‘wrapped’ objects for patchline updates whichever end of the cable they are.