FrameLib: Alpha 2.1 Documentation

Alex Harker

Generated on 8/6/2018 at 13:58

Contents

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 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. The 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 paraemters....

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 a 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 primarily 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. It's also possible to use tagged frames to concatenate information into a form which is easier to deal with until a point in the patch where one wishes to split the information apart again.

2.4   Multistream Expansion

Technically, the objects hosted in Max objects carry not single frames, but multistream 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 multistream inputs by internally expanding to the maximum stream count. Where there is a mismatch between different inputs the lower stream count inputs 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).

You can specify the minimum number of streams regardless of the inputs by making the first argument to the object something like this:

$=N$ where N is the minimum number of streams.

For audio objects this will determine how many copies of the audio ins/outs you see, so it can be used for multichannel IO using a single max object.

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.

fl.interval~   trigger regular frame - for irregular frames vary the interval over time.

fl.audiotrigger~   trigger frames using MSP non-zero values

fl.perblock~   trigger one frame per host audio block

fl.future~   schedule frames at specific points in the future

fl.once~   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:
        /parameter_name value(s) ...
within the object box. 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.tag~* object to tag frames before sending them to the parameter input of an 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. This is:

[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 send 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)

3.8   Code Export

You can now get C++ code from a Max patch of FrameLib objects. For the time being this is not fully documented, but it can be done using the **export** message, which takes a class name (a symbol) and a path to write files to (another symbol). Calling this will create a C++ class that hosts its own unique FrameLib network (it can't be connected to other FrameLib) networks) and can be tied in to some other audio block-based environment. The host should create the object, call the reset method when relevant, and call the process method in the audio thread at the correct time. The object can also be polled for its number of inputs/outputs. The build should be linked against libframelib_objects.a

Slightly more complexity is required for dealing with either buffers or sending values to/from the host. This is done via the FrameLib_Proxy class, which can be extended for various different scenarios of usage.

If you are interested in more detail on this, currently the best way to find out is to read the code on github, or email me.

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

Objects in bold are yet to be finalised by review.

5.1 Schedulers

fl.audiotrigger~	*fl.perblock~*	*fl.future~*	*fl.interval~*
fl.once~			

5.2 Unary

fl.abs~	*fl.acosh~*	*fl.acos~*	*fl.asinh~*
fl.asin~	*fl.atanh~*	*fl.atan~*	*fl.cbrt~*
fl.ceil~	*fl.cosh~*	*fl.cos~*	*fl.erfc~*
fl.erf~	*fl.exp2~*	*fl.exp~*	*fl.floor~*
fl.log10~	*fl.log2~*	*fl.log~*	*fl.not~*
fl.round~	*fl.sinh~*	*fl.sin~*	*fl.sqrt~*
fl.tanh~	*fl.tan~*	*fl.trunc~*	

5.3 Binary

fl.and~	*fl.atan2~*	*fl.copysign~*	*fl.diff~*
fl.divide~	*fl.equals~*	*fl.greaterthaneq~*	*fl.greaterthan~*
fl.hypot~	*fl.lessthaneq~*	*fl.lessthan~*	*fl.max~*
fl.minus~	*fl.min~*	*fl.modulo~*	*fl.notequals~*
fl.or~	*fl.plus~*	*fl.pow~*	*fl.times~*

5.4 Ternary

fl.clip~	*fl.fold~*	*fl.wrap~*

5.5 Complex Unary

fl.cartopol~	*fl.poltocar~*	*fl.complexconj~*	*fl.complexcosh~*
fl.complexcos~	*fl.complexexp~*	*fl.complexlog10~*	*fl.complexlog~*
fl.complextanh~	*fl.complextan~*	*fl.complexsinh~*	*fl.complexsin~*
fl.complexsqrt~			

5.6 Complex Binary

fl.complexdivide~	*fl.complexminus~*	*fl.complexmultiply~*	*fl.complexplus~*
fl.complexpow~			

5.7   Vector

fl.accumpoint~	**fl.argmin~**	**fl.argmax~**	**fl.centroid~**
fl.chop~	**fl.crest~**	**fl.flatness~**	**fl.geometricmean~**
fl.join~	**fl.kurtosis~**	**fl.length~**	**fl.mean~**
fl.medianfilter~	**fl.nanfilter~**	**fl.nonzero~**	**fl.pad~**
fl.peaks~	**fl.percentile~**	**fl.product~**	**fl.reverse~**
fl.rms~	**fl.shift~**	**fl.skewness~**	**fl.sort~**
fl.split~	**fl.spread~**	**fl.standarddeviation~**	**fl.subframe~**
fl.sum~	**fl.vmax~**	**fl.vmin~**	

5.8   Generators

fl.gaussian~	fl.ramp~	fl.random~	fl.uniform~

5.9   IO

fl.sink~	fl.source~	fl.trace~

5.10   Mapping

fl.constant~	**fl.convert~**	**fl.lookup~**	**fl.map~**
fl.samplerate~			

5.11   Host Communication

fl.frommax~	**fl.tomax~**

5.12   Buffers

fl.read~

5.13   Parameters

fl.combinetags~	fl.filtertags~	**fl.getparam~**	fl.tag~
fl.untag~			

5.14   Routing

fl.dispatch~	fl.route~	fl.select~

5.15   Streaming

fl.pack~	fl.streamid~	fl.unpack~

5.16   Spectral

fl.convolve~	**fl.correlate~**	**fl.fft~**	**fl.ifft~**
fl.multiaper~	**fl.window~**		

5.17   Time Smoothing

fl.ewma~   *fl.ewmsd~*   *fl.framedelta~*   *fl.lag~*

fl.timestddev~   *fl.timemean~*   *fl.timemedian~*

5.18   Storage

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

5.19   Timing

fl.now~   *fl.ticks~*   *fl.timer~*   *fl.timedelta~*

5.20   Filters

fl.0dfsvf~   *fl.onepole~*   *fl.onepolezero~*   *fl.resonant~*

fl.sallenkey~

5.21   Spatial

fl.coordinatesystem~   *fl.spatial~*

6   Alpha Aspects of FrameLib

N.B. Please note that objects not yet reviewed are more likely to change - the object list shows which objects are still due for review.

- 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

7.1 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.

7.2 Changes in Alpha 0.3

- Fixed testing for no live inputs to prevent crashes with audio-synced objects.

- [#1 and #14] Fixed *fl.medianfilter~* width clipping to prevent crashes.

- [#11] Updated *fl.fft~* and *fl.ifft~* to allow complex/full spectrum FFTs and to use the new HISSTools FFT code.

- [#12] Fixed binary templates to prevent overwrites in wrap mode with mismatched input lengths.

- [#15] Fixed *fl.source~* to allow dynamic changing of the size.

- 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.

7.3 Changes in Alpha 0.4

- [#2] Feedback loops no longer cause crashes.

7.4 Changes in Alpha 0.5

- [#18] Fixed table reading code to account correctly for the frame size.

- [#19] Fixed crashes with empty frames into vector template objects.

- [#20] Fixed crashing with negative values into *fl.peaks~*. .

- [#21] Added the *fl.tomax~* object. .

- Rebuilt the connection system to be shared between object types.

7.5   Changes in Alpha 0.6

- [#7] Objects now have channel awareness and shared memory can be channel specific.

- [#13] Added ordering connections that can take multiple connections for ordering purposes only, and also can be generated automagically.

- [#22] Fixed crash with incorrect *fl.ifft*~ input size tests.

- [#26] Fixed crashes with over length lists and *fl.tomax*~.

- [#26] Fixed crashes with two inputs of length 1 into *fl.convolve*~.

- More stringent checks for correct ordering and issues of scheduling.

- Significant reworking of the framework for ease of use and improved design.

- Move towards slashes for parameter entry in the max wrapper (demos changes).

7.6   Changes in Alpha 0.7

- [#27] Corrected median filter demo typo.

- [#28] Fixed issue with memory pool deletion on freeing of the global object.

- Fixed double reporting of patching errors whilst audio is on.

- Internal connection system updates and corrections (issues were currently non-visible).

7.7   Changes in Alpha 0.8

- #29 Removed support for multiple methods for entering parameters/inputs.

- #29 and #31 Updated binary object parameter naming scheme.

- #31 Added init folder to allow aliasing to symbol-based names for binary ops.

- #32 Added support for better multichannel control, including multichannel audio in and out (plus a demo patch).

- #32 Minor info string updates for binary objects.

- Added new objects (*fl.not*~, *fl.and*~, *fl.or*~, *fl.modulo*~, *fl.channel*~).

7.8   Changes in Alpha 0.9

- #36 Changed *fl.select*~ parameter naming ("num_ins").

- #38 Fixed *fl.read*~ crash with non-existent buffer or buffer names including numbers.

- Added new object *fl.route*~.

- Added new demos for the *fl.route*~ object.

7.9 Changes in Alpha 1.0

7.9.1 Bug fixes

- #4 Probing on audio outs now works.

- #41 and #43 Added a parameter input to *fl.read*∼ and *fl.subframe*∼ (as well as many other objects).

- #42 *fl.modulo*∼ now works correctly.

- #45 Fixed crash with *fl.store*∼.

- *fl.convolve*∼ output is now scaled correctly.

- Crash fixes for the connection system and further reworking (dependencies now handled by Max API).

- Corrected mode setting for *fl.lookup*∼.

- Improved documentation for the *fl.atan2*∼ object.

7.9.2 New objects

- #34 New timing objects *fl.timer*∼, flobjecttimedelta and *fl.ticks*∼ (by Francesco Cameli).

- #35 New parameter objects *fl.untag*∼, *fl.filtertags*∼, *fl.combinetags*∼ and *fl.getparam*∼.

- #36 New routing object *fl.dispatch*∼.

- New generator object *fl.uniform*∼.

- New vector object *fl.nonzero*∼ (by Francesco Cameli).

7.9.3 Renamed objects

- Object *fl.count*∼ is renamed as *fl.ramp*∼.

- Object *fl.channel*∼ is renamed as *fl.streamid*∼.

- Object *fl.setparam*∼ is renamed as *fl.tag*∼.

7.9.4 New object features

- *fl.ramp*∼ supports different output scalings for ease.

- *fl.trace*∼ now supports different modes for which sample(s) to trace.

- All IO objects support setting the delay at input/output.

7.9.5 Parameter changes

- Parameter naming consistent for setting number of ins/out ("num_ins" and "num_outs").

- Parameter name is always "length" for output length - internal buffers use "buffer_size".

- IO objects have revised parameters/arguments and support units for length.

- Generator objects have revised parameters/arguments and support units for length.

- #35 Parameter objects now don't need an argument for the number of tags.

- #36 Routing objects now index inputs/outputs from 1 (for consistency) and default to off/non-routed.

- Storage objects can now deal with both vectors and tagged frames correctly.

7.9.6   Other changes

- #32 Multi-stream expansion syntax changed to =.

- Binary size reduced.

- All demos updated to reflect object changes.

- New demos added.

7.10   Changes in Alpha 2.0

7.10.1   Bug fixes

- #48 Compilation no longer requires AVX.

- Corrected info strings for *fl.* ~*, *fl.trace~*, *fl.constant~* and *fl.medianfilter~*

- Corrected *fl.length~* to report zero length frames.

- Corrected *fl.percentile~* to report the correct values.

- Corrected *fl.fft~* to calculate FFTs with complex inputs correctly.

- Corrected *fl.ramp~* scaling.

- Corrected *fl.medianfilter~* edge behaviour.

- Corrected variance calculations for *fl.fl.starndarddeviation~* etc.

- #4 Probing the outputs of FrameLib objects in Max now works.

- #60 Fixed hang with *fl.gaussian~* and zero length inputs.

- #61 Fixed crash with *fl.shift~* with large shifts in padding mode.

- FrameLib now works correctly in *poly ~* and similar objects, even when nested.

7.10.2   New objects

- New reverse object *fl.reverse~* (by James Bradbury - #51).

- New objects to return the position of min/max values - *fl.argmin~* and *fl.argmax~* (by Owen Green).

- New nan filtering object *fl.nanfilter~* (by Owen Green).

- New objects for constraining values *fl.clip~*, *fl.wrap~* and *fl.fold~* (by Owen Green).

- New gaussian random number generation object *fl.gaussian~*.

- New correlation object *fl.correlate~*.

- New object for calculating the sample at which a given percentage of the sum is reached *fl.accumpoint~*.

- New complex math objects (numerous but including *fl.cartopol~* and *fl.poltocar~* as raised in #10).

- New time smoothing objects for frame delays, smoothing and statistics over time in parallel values per vector (e.g. *fl.timemean~* and *fl.lag~* as raised in #8).

7.10.3 New object features

- *fl.convolve~* and *fl.correlate~* supports real and complex inputs as well as a number of edge modes.

- *fl.lookup~* now supports different kinds of input scaling via the "scale" parameter.

- *fl.sort~* now supports output in the form of indices, as well as by value (see the "indices_mode" parameter).

- *fl.constant~* can now return an epsilon value (thanks to Owen Green).

- #52 *fl.read~* now supports changing the buffer in realtime.

7.10.4 Other changes

- Corrected trigger inputs on the "clever" granular synthesis demo.

- Corrected buffer length parameter for the *fl.sink~* object in the "clever" granular synthesis demo.

- Added new demos for 2D median filtering and stochastic freeze.

- FrameLib is now compilable for pd (not quite ready for testing yet).

- Added code export feature that can be used to auto-generate C++ code from a network in Max/pd.

7.11 Changes in Alpha 2.1

7.11.1 Bug fixes

- #66 Corrected naming of not equals object.

- Ternary objects now behave predictably for all input values.

7.11.2 New objects

- #9 *fl.expr~* and *fl.complexexpr~* allow expressions in a single object.

7.11.3 Other changes

- #68 FrameLib is now compiled with AVX1.0 but not AVX2.0 due to incompatibility with reasonably recent computers.

- #63 Corrected typos in demos and user docs.