Cloning Max/MSP Objects: a Proposal for the Upgrade of Cyclone

Alexandre Torres Porres
EL Locus Solus
São Paulo-SP Brazil
porres@gmail.com

Derek Kwan
Fresno, CA USA
derek.x.kwan@gmail.com

Matthew Barber
Rochester, NY USA
brbrofsvl@gmail.com

Abstract

The Max/MSP compatibility library Cyclone has been updated, extending compatibility to Max versions 5, 6 and 7 over the current canonical Cyclone version's compatibility with Max version 4. Primary development goals have included updating existing objects to reflect features introduced in newer versions of Max, cloning Max objects missing from Cyclone, fixing bugs in existing code, and addressing errata found in Max objects and documentation. Secondary goals have involved an upgrade to documentation, streamlining the codebase, and improving the build system. Community discussion will resolve the project's future as either an update or fork of the current Cyclone.

Keywords

Cyclone, Max/MSP, Externals, Library, Update

1 Introduction

The Cyclone library is a set of externals that clones object classes from Cycling '74's Max/MSP software for Pure Data. Cyclone expands Pd with a large set of externals and provides some level of compatibility between Pd and Max, helping users of both systems in the development of equivalent or similar patches, and giving Max users a convenient gateway to Pd. Original development of Cyclone brought compatibility with Max version 4. We present a significant update to Cyclone which brings compatibility up to recent Max versions (7.3.0 at the time of this writing) by adding new functionalities introduced in newer Max versions to Cyclone's classes, fixing bugs, and writing new documentation.

2 A Brief History of Cyclone

Putting our update in its proper coding and social contexts requires a history of Cyclone.

2.1 Maintainers and Versions

The first iteration of Cyclone was begun in April 2002 by Krzysztof Czaja, when Max was at version 4.0. [1] The earliest experimental alpha release, cyclone-0.1-alpha1, was introduced to the Pd community in November 2002. [2] It had 60 control (Max) objects, and 12 signal (MSP) objects, collectively named "hammer" and "sickle," respectively. Cyclone grew to 166 cloned objects by 2005, but was still in alpha development (cyclone-0.1-alpha55).[1]

Cyclone could be loaded in several ways: in one, each object could be loaded from its own compiled binary. In another, `hammer`, a sub-library with all the control objects compiled into one file, and `sickle`, a similar sub-library with the signal objects could be loaded separately. Loading the `cyclone` sub-library would load `hammer` and `sickle`, as well as a few non-alphanumeric signal "bin-ops" like `>=~`, known in Cyclone as "nettles." Loading the sub-library `maxmode` would load `hammer`, `sickle`, and `cyclone`, along with a suite of "dummy" classes, which instantiated nonfunctional objects with the names of classes not yet cloned, so that – in principle – a Max patch could be opened from Pd. Finally, an executable binary `cyclist` was provided to convert binary Max patches into the equivalent text forms.

Hans-Christoph Steiner made Cyclone available in Pd-extended, and because Czaja ceased developing it in 2005, Steiner took over maintenance of Cyclone until 2013. The library was updated to cyclone-0.1-alpha56 for Pd-extended version 0.43.[2] All development on Pd-extended ceased in 2013, and Cyclone was left unmaintained as a result.

Fred Jan Kraan assumed maintenance of Cyclone in December 2014. He made cyclone-0.1-alpha57 available via the `deken` library manager plugin, and replaced it with the current canoni-

[1] Although Krzysztof Czaja's webpage no longer exists, a copy may be found at `http://fjkraan.home.xs4all.nl/digaud/puredata/cyclone/cyclone_site/cyclone.html`, which includes a downloadable source for cyclone-0.1-alpha54.

[2] Pd-extended releases may be found at `https://puredata.info/downloads/pd-extended/releases/` and at the sourceforge.net repository `https://sourceforge.net/projects/pure-data/files/pd-extended/`

cal version cyclone-0.2beta in December 2015. In February 2016, Kraan decided to abandon active development, but he still maintains the cyclone-0.2beta package. This version contains the original 166 cloned cyclone objects plus a `teeth~` object. The individual `hammer` and `sickle` libraries were no longer compiled by default. Kraan also fixed several bugs and revamped the documentation.

2.2 Cyclone Repositories

The original Cyclone library was a sub-library of the `miXed` library, which was developed by Cyclone's original author Krzysztof Czaja.[3] Kraan's forked git repository hosts versions cyclone-0.1-alpha57 and cyclone-0.2beta, and employs Katja Vetter's pd-lib-builder.[4]

2.2.1 Our Repository

In February 2016, as a response to Fred Jan Kraan's decision to cease development and provide maintenance only, Porres forked cyclone-0.2-beta1 to a new git repository to continue Cyclone development.[5] Our branch is in active development, independent of the original developer and subsequent maintainers.[6] Our intent is to continue work on Cyclone without diverging from its original goal and scope. Krzysztof Czaja could not be reached to discuss our plans for Cyclone development, but Steiner created ground rules for Cyclone development on the Pd-list:

> About maintaining cyclone, I think a reorg would be great, and further maintenance as well. If you want to do whatever you want with it, then just make a fork and work on it as a new name. If you want to stick to cyclone's central goal of Max/MSP compatibility, then keep working on it as cyclone. But please do not work on cyclone and break the Max/MSP compatibility. [3]

Because the Cyclone library has a very clearly defined scope – it consists only of cloned objects from Max/MSP – we have been able to remain faithful to its central goal. Thus our intent is to continue developing this library as Cyclone rather than as a fork with a new name. We do not wish to impose any personal view on or control over the project; on the contrary, we consider Cyclone to belong ultimately to the Pd community and not to any particular maintainer. We are therefore eager to collaborate with anyone who wishes to help with fixing bugs and coding new objects. Before releasing an official version of our work, we wish to discuss the future of Cyclone with the Pd community. With community agreement we plan to release our project as cyclone-0.3, with an alpha test version ready to demo at PdCon16.

3 Cyclone 0.3

Cyclone was originally developed in the Max 4 era, and Cycling '74 has since introduced new functionalities from Max versions 5 to 7. Cyclone was compliant only with Max 4 through version alpha56, but when Kraan took over he was open to including features from Max 5. The current canonical version of Cyclone (0.2beta) is still for the most part compliant only with Max 4, with a few exceptions (for instance, the `clear` method of `delay~` was new with Max 5 and is implemented in cyclone-0.2beta).

Our primary goals for Cyclone version 0.3 have been updating the existing codebase to reflect new features from Max versions 5-7 (up to 7.3.0 as of this writing), cloning important objects currently missing from Cyclone, fixing bugs in Cyclone code, and addressing bugs and errata in Max objects and documentation. In order to make the project more accessible for users and developers, we have completely rewritten the Cyclone documentation, restructured the codebase, and improved the build process.

3.1 Updating Cyclone to Max 7

Updating existing Cyclone objects to Max 7 compatibility and writing documentation to reflect this has received by far our most intense focus and hardest work. A careful analysis showed that 54 objects required an update to be compliant with

[3]There are two `miXed` repositories, a sourceforge.net site, `https://svn.code.sf.net/p/pure-data/svn/trunk/externals/miXed/`, and a migrated git repository `https://git.puredata.info/cgit/svn2git/libraries/miXed.git/`

[4]Kraan's git repository is `https://github.com/electrickery/pd-miXedSon` (note that although it is called pd-miXedSon, only the Cyclone library has been forked). Vetter's pd-lib-builder is available at `https://github.com/pure-data/pd-lib-builder`.

[5]Our branch is `https://github.com/porres/pd-cyclone`

[6]Kraan graciously links to our branch as the one in active development.

recent Max versions.[7] This work is nearly complete as of this writing.[8]

Cyclone's relationship to Max 4 is complicated. Max 4 was quite long lived, with updates from 2001 to 2007, which covered versions 4.0 to 4.6 (Max 5 was released in 2008). Since Cyclone was first released in 2002 with compatibility to Max 4.0, changes in Max after version 4.0 were not always reflected in the Cyclone code. For instance, the second argument to `coll`, which prevents the object from searching for a file with the symbol given in the first argument, was introduced in Max 4.0.8, but was never implemented in Cyclone.

Moreover, some features from Max 4.0 never made it into Cyclone objects (e.g. the missing "symout" argument to `sprintf`), and a number of bugs persist. We have treated the various missing features from Max 4.0 to 4.6 as bugs, since cyclone-0.1-alpha56 contained features from Max 4.6 and seemed to be incomplete on its own terms. Therefore we have not counted them among the 54 objects requiring update from Max 4 to Max 5+ compatibility. Part of this distinction is academic, since most of the objects missing features from Max 4 also needed updates to bring compatibility to Max 5+ (exceptions include `funbuff`, `mousestate`, `substitute`, and `slide~`).

To put this in perspective, we note that 54 objects is just under one-third of the 166 originally cloned objects, so the majority of them have not changed in Max from versions 4.6 to 7.3. Max/MSP objects have tended to reach a level of stability and maturity after time, but sometimes old bugs and kludges have also tended to be retained in mature objects (see **3.2.4 Inconsistencies** below).

Fortunately, nothing has been introduced in newer versions of Max that makes an update of Cyclone impractical. In fact, the major feature additions to Max have come in the form of large packages such as Jitter, Gen, and JavaScript capability, which can safely be left out of Cyclone. Most of the major changes to objects between Max 4.6 and 7.3 occurred in versions 5 and 6. `scope~` is the only object currently cloned in Cyclone to change in Max from versions 6 to 7 (this was merely a difference in default color scheme); only `midiparse` and `midiformat` received updates between Max 7.0 and Max 7.3.0, requiring updates in Cyclone.

3.1.1 Attributes

Some updates to Max cannot be as easily addressed as the addition of methods to existing objects, for example. Max "attributes" illustrate how our work brings some Max semantics into Cyclone and make them compatible with Pd. Attributes in Max function similarly to property settings and option flags in Pd. Although present in a few object classes from Max 4.5, attributes began to proliferate among Max/MSP's object classes in Max versions 5 and 6. [4] Property windows exist for GUI objects in Pd, and some classes (e.g. `declare` and `sigmund~`) take flags to control how the object operates. In current Max versions, many object settings can be controlled with attributes, using the special attribute syntax `[object @attribute-name <setting>]`.

In addition, every object has a set of attributes called "common box attributes," which set the look and feel of the object box either via `@attribute` flags to the object or a graphical menu in the inspector. Since our purpose is only to clone the function of Max classes and not their appearance, we have opted not to include support for common box attributes. While Pd's `-flag` options are syntactically identical to Max's `@attribute` options, we decided to use Max-style attributes rather than Pd-style flags for this kind of option setting in Cyclone. It is faithful to the Max compatibility goal and it provides Max users with a familiar environment.

In Max, once an object is instantiated, attributes can be set dynamically in three additional ways: first, via the inspector, second with a special attribute-editing object `attrui`, and third by sending the object a message with the attribute name and its argument(s). Pd does not have an inspector, but rather calls dedicated properties windows for GUI objects; therefore for code simplicity we plan to make properties windows only for GUI objects like we did with `scope~`, leaving the other objects with only `@attribute` flags. The `attrui` object inspects an object it is connected to for attributes and allows the user to set them from the patch. While such an object is possible in principle for Pd, we have decided to implement dynamic attribute setting via message passing only, saving any development of `attrui` for the future.

[7] An outlier is the `comment` object, which we are also updating. However, due to differences in GUI toolkits and platforms will likely never behave in exactly the same way as Max's, so it is not listed among these 54.

[8] Current progress, as well as a complete list of objects affected by our work, may be found at `https://github.com/porres/pd-cyclone/wiki/cyclone-0.3-changlelog`

```
while(argc > 0)
{
  if(argv -> a_type == A_FLOAT)
  {
    t_float argval = atom_getfloatarg(0, argc, argv);
    switch(argnum)
    {
      case 0:
        tripeak = argval;
        break;
      default:
        break;
    }
    argnum++;
    argc--;
    argv++;
  }
  else if (argv -> a_type == A_SYMBOL)
  {
    t_symbol *curarg=atom_getsymbolarg(0, argc, argv);
    if(strcmp(curarg->s_name, "@lo")==0)
    {
      if(argc >= 2)
      {
        trilo = atom_getfloatarg(1, argc, argv);
        argc-=2;
        argv+=2;
      }
      else goto errstate;
    }
    else if(strcmp(curarg->s_name, "@hi")==0)
    {
      if(argc >= 2)
      {
        trihi = atom_getfloatarg(1, argc, argv);
        argc-=2;
        argv+=2;
      }
      else goto errstate;
    }
    else goto errstate;
  }
  else goto errstate;
}
```

Figure 1. The attribute parsing loop from `triangle~`.

Introduction of attributes necessitates parsing an object's arguments as a list. If an object has more than one settable attribute, the `@attribute` flags may be called in any order, which requires a flexible constructor. As with Pd vanilla objects that allow flag options, elements of the argument list are parsed one-by-one, comparing any symbols to the symbols in the list of `@attribute` possibilities for that object. If the symbol does not match any attributes, then it may be considered a normal creation argument, such as the name of an array. If neither condition holds, or if the requisite number of arguments to the `@attribute` are not present, the constructor lands in an error state and the object does not instantiate. **Figure 1** is an example of an attribute parsing loop from `triangle~`, which has two possible attributes, `@lo` and `@hi`.

3.1.2 "Magic"

The difference between Max's patching syntax and Pd's is a common obstacle for Max users who are new to Pd. These differences make things difficult for a Max compatibility library. Some of them are intractable on the Pd side without making changes to the Pd core; for instance, Pd message fanouts require `trigger` to operate in proper order while Max controls order by spatial placement of objects. Others demand a hybrid approach; for instance Max does not share Pd's commitment to deterministic depth-first processing. For instance, `coll` can load large files in the background while other control and signal operations continue; in Cyclone, `coll` can be made to be deterministic if desired, or to use a threaded file-loading function that breaks determinism.[9]

There are also differences in signal inlet behavior between Max and Pd that can be addressed from within the external class. In Pd's tilde objects, secondary signal inlets contain a float field that can be set by an incoming float. If the inlet has an incoming signal connection it will ignore that float field. If it does not, Pd writes the value of that float field to its inlet's signal vector, thus promoting floats to signals. The object's behavior is the same in each case. In Max/MSP, floats are usually promoted to signals, but sometimes they are ignored or used to set specific parameters whether or not a signal is connected. Likewise some MSP objects have different internal behaviors depending on whether a signal is connected to one or more of its inlets, regardless of whether floats are promoted to signals.

This discrepancy is difficult to resolve because Pd and Max users have different expectations, and Cyclone has to fulfill commitments to both. Past Cyclone versions have favored matching the behavior of Max objects' inlets whenever the Pd API makes it possible, and so we have decided to continue this practice and extend it. Cyclone contains a collection of shared "unstable" modules which work by adapting Pd's API in novel and sometimes obscure ways.

The three primary modules involved are `forky`, which contains sections of conditionally compiled code that preserve functionality when Cyclone is compiled against different versions of Pd; `fragile`, which contains functions that depend on specific

[9] In principle the latter is not much different from employing a random `delay` in a message chain.

```
int forky_hasfeeders(t_object *x, t_glist *glist,
    int inno, t_symbol *outsym)
{
    t_linetraverser t;
    linetraverser_start(&t, glist);
    while (linetraverser_next(&t))
    {
        if (t.tr_ob2 == x && t.tr_inno == inno
#if FORKY_VERSION >= 36
        && (!outsym ||
            outsym == outlet_getsymbol(t.tr_outlet))
#endif
        )
        return (1);
    }
    return (0);
}

/*-----------------------------------------------------*/

static void scope_dsp(t_scope *x, t_signal **sp)
{
    x->x_ksr = sp[0]->s_sr * 0.001;
    int xfeeder, yfeeder;
    xfeeder = forky_hasfeeders((t_object *)x,
        x->x_glist, 0, &s_signal);
    yfeeder = forky_hasfeeders((t_object *)x,
        x->x_glist, 1, &s_signal);
    scope_setxymode(x, xfeeder + 2 * yfeeder);
    dsp_add(scope_perform, 4, x,
        sp[0]->s_n, sp[0]->s_vec, sp[1]->s_vec);
}
```

Figure 2. `forky_hasfeeders()`, **and its call in** `scope~`.

implementation details of Pd (e.g. from `m_imp.h`) and which are likely to break in new versions of Pd; and `fringe`, which contains functions deemed likely to be included in Pd's official API in the future.[10] In personal correspondence about the project, we have begun calling this collection of novel functions "magic tricks."

A good illustration of the magic tricks and how they are applied is the signal visualization class `scope~`. This object has two signal inlets and behaves differently depending on how the inlets are fed.[11] When only the left inlet has signal input, the signal is drawn with time plotted on the horizontal axis and amplitude on the vertical; this arrangement is reversed when only the right inlet has a signal, with time on the vertical axis and amplitude on the horizontal. When both are connected with a signal, the object goes into "X-Y mode," and plots the signals parametrically on a Cartesian plane. When neither inlet is connected, the object draws a flat horizontal zero-amplitude line. Furthermore, neither inlet promotes floats to signals: floats in the right inlet set the number of signal points displayed in the display buffer, and floats in the left inlet set the number of signal samples used to draw each point. The left inlet also accepts a number of method messages.

In order to alter the behavior of an object based on its signal input configuration, Cyclone classes use the function `forky_hasfeeders()` (see **Figure 2**). When called from within an object, this function traverses the object's canvas connections until it finds the object and the relevant inlet, and then checks whether it is connected, and if so whether that connection is a signal connection. It is called in the `dsp()` routine, and can be used to set options or load different `perform()` functions based on the return value.

Keeping the two signal inlets from promoting floats to signals is more difficult, and the procedures involved are newly introduced into Cyclone classes. These procedures differ for left and right inlets. Left inlets in Pd have an underlying infrastructure that automatically allows them to receive both signals and messages, which is usually accessed through the `CLASS_MAINSIGNALIN()` macro. In the usual case, incoming floats to the left inlet sets a float member of the class's struct, which are internally promoted to signals whenever no signal connection is attached. This can be overridden by giving the class both a signal and a float method, as in this example from `scope~`:

```
class_addmethod(scope_class,nullfn,gensym("signal"),0);
class_addfloat(scope_class,(t_method)scope_float);
```

Incoming floats now call `scope_float()` instead of the default float method.

Doing the same with secondary inlets is more difficult because float inputs do not automatically call a class method, but rather set a float field in the inlet's struct. Direct access to this field is usually hidden, but the `obj_findsignalscalar()` routine from Pd's `m_obj.c` returns that field's address.[12] The class's `perform()` routine(s) can then poll that field for changes every tick and call a method if a change is detected. Here is how it works in `scope~`:

[10] This organization is both pragmatic and problematic, and is undergoing restructuring. See **3.2.1 Code Restructuring** below for more details.

[11] The following only applies when DSP is on.

[12] Cyclone's `fragile` module contains a similar routine, `fragile_inlet_signalscalar()`. We might phase this out because it requires maintaining a copy of the `inletunion` and `_inlet` declarations in `m_obj.c`, whereas using `obj_findsignalscalar()` only requires declaring it as an `EXTERN` with the proper arguments.

```
int bufsize = (int)*x->x_signalscalar;
if (bufsize != x->x_bufsize)
   scope_bufsize(x, bufsize);
```

3.1.3 Other Significant Revisions

We have made a number of especially significant revisions to some classes in the process of moving to Cyclone version 0.3. The following is a brief catalog of salient examples.

- Previous versions of `comb~` and `allpass~` all had an error in the difference equation that required a complete rewrite of their `perform()` routines, including the addition of secondary delay buffers. We added a new `teeth~` object based on `comb~` to replace a former abstraction.
- `cycle~` now has an internal 16384-point, fully symmetric cosine table so that, unlike with `osc~` and previous iterations of `cycle~`, frequency modulation is stable over time.
- We have implemented the extra interpolators that were added to `wave~` in Max 6. Since Pd's cubic Lagrange interpolator is not among them, but was included in previous versions of Cyclone, we have retained it as an option.[13]
- `delay~` now accepts signals to control samples of delay, with cubic interpolation.
- The bitwise signal operators `bitand~`, `bitor~`, and `bitxor~` have been revised to fix their second inlets (which set the objects' bitmasks) using the procedures described in the previous section. All of the type punning required by these objects has been rewritten for stability, replacing pointer casts with unions. Denormal output values are set to zero as they are in Max.
- `train~` now allows pulses of width 0 (resulting in single-sample widths) and width 1 (resulting in widths of one sample less than an entire cycle). The phase behavior has also been altered to respond correctly when changed.
- `fromsymbol` now allows any single- or multi-character string to act as delimiter.
- `funnel` has been almost completely rewritten so that lists are stored properly at each inlet and output correctly upon receiving a `bang` (previously only the first element of a list was stored and output on `bang`).
- `sustain` has been rewritten to accommodate two additional modes for handling repeated note-on messages, which were introduced after Max 4.
- `scope~` has undergone major surgery. It now supports the "Y-only" mode and the "alternate drawstyle" options. It has new methods allowing `rgb` values to be set according to a 0.0 to 1.0 float range. The "X-Y" mode has been simplified and its performance improved. Several bugs were fixed, including a crasher bug on x86_64. Finally, it has a new Pd-style properties window.

3.2 Beyond Max Compatibility

Our primary goal has been bringing existing Cyclone code up to full compatibility with Max 5+, but we have made various other improvements along the way. Besides updating the documentation, we have restructured the codebase, created a number of new cloned objects, and fixed bugs, many of them longstanding. We also addressed some bugs in the Max objects themselves.

3.2.1 Code Restructuring

Over the course of the update it became clear that the Cyclone codebase was in need of restructuring and streamlining. Fred Jan Kraan initiated this process in alpha57 and 0.2beta by importing pd-lib-builder and adjusting the build targets. As of 0.2beta (the current version), the sub-libraries `hammer`, `sickle`, and `maxmode` are not compiled by default. The `cyclone` sub-library has been renamed `nettles`, and it only loads the non-alphanumeric bin-ops (and not all of the control and signal objects).

We have taken the restructuring process several steps further. Given the increasingly wider divergence of Max and Pd since the Max 4 era, any effort to emulate Max as deeply as earlier Cyclone attempts seems forlorn. We therefore no longer keep `cyclist` or the legacy sub-libraries as build targets, but keep their code in a maintenance directory. We did, however, restore the `cyclone` sub-library, which loads the bin-ops and serves as a space for future implementation of some of Max's syntactic sugar (e.g. the `zl.mode` syntax). Since the sub-libraries are no longer in operation, all of the source files for the object classes have been moved from "hammer" and "sickle" directories to "control" and "signal" directories. This also helps newcomers by making the purpose of the directory structure clear.

The examples in previous sections illustrate the extent to which Cyclone relies heavily on code modules shared among the object classes. Many of the functions in these modules serve as thin

[13] The MSP `wave~`'s new interpolators are identical with the ones found in a 1999 web article by Paul Bourke, and they are included in the same order they appear in the paper. The Max documentation contains no attribution; Bourke's article is here: `http://paulbourke.net/miscellaneous/interpolation/`.

wrappers over the standard Pd API. The two most important are the `sic` and `arsic` (i.e. "sickle" and "array sickle") modules, which create `t_sic` and `t_arsic` types inheriting from Pd's `t_object`. These modules also implement custom creation methods such as `arsic_new()` and `sic_inlet()`, and provide routines for use in all signal object classes (`sic`) and all object classes that access Pd arrays (`arsic`). We have begun editing object classes to remove this cruft. By standardizing Cyclone to comport with the Pd API, we have rendered the codebase more transparent and made our new contributions much easier to code, especially for attributes that affect inlet instantiation and the "magic" code.

Eliminating dependence on the `sic` module has been relatively easy, amounting to replacing the wrapper functions with those from the standard Pd API. Disentangling from the `arsic` module has been much more difficult, because `arsic` emulates MSP's multichannel `buffer~` by allowing signal classes to read from and write to multiple Pd arrays (which are collected according to a naming convention). `arsic` also depends on the "vector of floats" submodule `vefl`, which has custom functions for gaining access to Pd array contents.

The necessary functions from `vefl` and `arsic` have been collected into a new "Cyclone buffer" module `cybuf`. This module introduces a new type, `t_cybuf`, which is similar to `t_arsic` except it has no `t_object` member. An object class that depends on `cybuf` can now keep its own `t_object` member, which used to be replaced with `t_arsic`. Now that the array functions in `cybuf` are separated from the standard Pd object creation methods, they can be readily replaced or supplemented if a proper `buffer~` class is developed, or if multichannel arrays are ever introduced in vanilla Pd.

As mentioned above, the code in the `forky`, `fragile`, and `fringe` modules is not collected according to similar function, but rather similar maintenance status vis-à-vis Pd's API. From a pragmatic standpoint, this does make maintenance somewhat easier because there is less disruption when there is a change in Pd. On the other hand, it makes development more difficult, especially for new contributors. One example that proves this point is inlet and outlet handling. In **3.1.2 "Magic"** we discussed the `forky_hasfeeders()` function, which returns the connection status of an inlet. There is a corresponding function that does the same for outlets, but it is found in the `fragile` module instead – `fragile_outlet_connections()`. There are many similar examples. We plan to reorganize these modules according to function, and in the process eliminate code that provides compatibility with very old versions of Pd. We hope that these and other improvements to the code structure will encourage participation from other contributors, who may not have wanted to learn an unnecessary and complicated wrapper API.

3.2.2 New Object Classes

Cyclone 0.3 introduces 45 new object classes that were not present in previous versions.

13 control classes: `acosh asinh atanh atodb dbtoa join loadmess pak pong rdiv rminus round scale`

32 signal classes: `atodb~ biquad~ bitsafe~ cascade~ cross~ dbtoa~ degrade~ downsamp~ equals~ filtercoeff~ freqshift~ gate~ greaterthan~ greaterthaneq~ hilbert~ lessthan~ lessthaneq~ modulo~ notequals~ number~ phaseshift~ plusequals~ rdiv~ rect~ rminus~ round~ saw~ scale~ selector~ thresh~ tri~ trunc~`

The bandlimited oscillators `rect~`, `saw~`, and `tri~` as well as the signal number box `number~` currently exist in Cyclone 0.3 as abstractions, but we are planning to implement them as externals in future versions.

3.2.3 Documentation

Czaja originally released Cyclone with no documentation. Many of the changes to Cyclone over the years were attempts to create good documentation. After careful scrutiny we found that much of the existing documentation had mistakes. Because we were already planning on a major documentation update for the new and upgraded objects, we made the decision to rewrite all of the documentation from scratch. This had the dual purpose of systematically testing every Cyclone object to ensure it complied with the specifications from the Max/MSP documentation.

In the process we found that the Max/MSP objects and documentation also had a number of bugs and inconsistencies, which made reverse engineering that much more difficult. Many objects had undocumented behaviors, so we had to make several decisions about whether a given undocumented behavior was a feature or a bug. We have

tried not to duplicate obvious bugs, and to thoroughly document the behaviors we have retained.

In some cases we have even rewritten the stated purpose of an object, where the Max documentation was either misleading or incomplete. For instance, the documentation for `wave~` bills it as a sample player, where in fact it is an all-purpose buffer reader that can be used for oscillating waveforms, waveshaping, etc. This major overhaul is still far from complete, however, and is likely to be what delays a transition from alpha to beta.

3.2.4 Inconsistencies

As discussed above, the many differences between Pd and Max have made cloning some features difficult. There are still several inconsistencies between some Max 7 objects and their Cyclone 0.3 counterparts. As of version 0.47.1, Pd lacks a multichannel array object, implemented in MSP as `buffer~`. The MSP objects `index~`, `peek~`, `buffir~`, `poke~`, `wave~`, `record~`, and `play~` all read from and/or write to `buffer~` objects. These objects in Cyclone 0.3 have been written to use the `cybuf` module. As mentioned above, the functions in `cybuf` can be replaced or supplemented by the introduction of multichannel arrays in Pd vanilla or a new `buffer~` clone. Backwards compatibility would probably indicate a hybrid approach here, with support for any and all of these options.

Another major inconsistency is that Pd lacks anything like Max's `transport` object, which provides a globally accessible scheduling clock and optional secondary clocks bound to symbol-defined names. The `transport` object is used to allow certain objects (e.g. `record~` and `delay~`) to define time intervals relative to a tempo rather than absolute terms. All of the classes which utilize `transport` in Max/MSP have been written without this functionality.

Due to differences in toolkit font rendering, the GUI object `comment` may never be fully compliant with Max. We invite developers who are proficient in tcl/tk to help make `comment` behave more consistently across platforms.

4 Roadmap

Here we discuss our vision for the future of Cyclone. We want to emphasize that because Cyclone belongs to the Pd community, all of the following is subject to revision according to the needs of the community.

4.1 Plans for New Code

In the near future, once we have reached a stage in development for a stable release, we plan to make Cyclone 0.3 available for all platforms via Pd's externals manager and in upcoming versions of Pd-l2ork and Purr Data. The source code will be available on GitHub as usual, and we hope also on `puredata.info`. As stated earlier, we welcome contributions from the Pd community in any form.

Despite our progress, Cyclone is far from complete; there are many Max/MSP object classes we plan on including in future releases. New GUI object classes and Max's more complicated control and signal object classes present the most exciting opportunities for future development.

Cyclone currently only has two GUI object classes (`comment` and `scope~`). Users making the transition from Max to Pd lament the dearth of GUI objects, so we hope to introduce more in future releases. Such classes include `radiogroup`, a one-dimensional grid of toggles; `matrixctrl`, a two-dimensional grid of toggles or knobs for use with `matrix~`; `rslider`, a slider which allows the user to select a number range; `kslider`, a visual piano keyboard representation; `multislider`, an object class similar to Pd's garrays, and `spectroscope~`, which displays a signal's spectrogram or sonogram. Any new development of GUI object classes should make as efficient use of tcl/tk as possible. A secondary consideration is porting to the nwjs toolkit used in Purr Data.

We have already mentioned `transport` and `buffer~` as potential new control and signal classes. The hash-table dictionary object `dict` would be an important contribution given the problematic accretion of features and bugs in Max's `coll`, which has similar functionality. We hope also to clone the signal object classes `groove~`, a user-friendly sample player, and `gizmo~`, which detects and shifts frequency peaks of an FFT analysis for pitch transposition. In any event, the Pd community's priorities ought to help shape future development.

4.2 Community Logistics

To conclude this paper it is necessary to discuss logistics of Pd community involvement going forward. First, some words about the legal ramifications of reverse engineering software are in order. Cyclone has always existed in a legal gray area. Although it employs names, syntax, and semantics from Max/MSP, the algorithms and pro-

cedures used in the code are the original work of the developers. It may also be problematic that it specifically purports to offer Max/MSP compatibility. In the unlikely event of legal trouble, we ought to have a contingency plan in place to salvage the library, and community input is welcome.

Second, there have already been a number of disagreements about the status and direction of this project, and a good-faith effort to resolve them are in the best interests of all involved. The primary disagreement has been about whether this project is best released as a new (forked) library with a new name, or as a new version of Cyclone taking over as the canonical development branch.

There are well-reasoned arguments for both positions. Here are some of the best reasons for releasing this update under a new name.[14] A supplemental library of new objects could be released under a new, similar name such as "Typhoon," and then users could install both libraries to get all the objects. If they do overlap, two non-identical libraries with the same name is sure to be confusing to users, while two libraries with similar goals but with different names would give users finer control over what they install and use. Forking projects to add new features under a different name is a common – even normative – development strategy in open-source software; after all, Pd has already had several forks of its own and users have managed to navigate this terrain. Finally, Cyclone's infrastructure is overly complicated, and a library of cloned objects might do better not to use it.

We respectfully disagree with much of this reasoning. It should be clear from the foregoing that our primary goal has been to bring the existing object classes from Cyclone up to date, not merely to supplement it with new classes. It would make some sense to release a separate library of new objects with a new name if it did not overlap in content with the old Cyclone, but name clashes resulting from the existence of different classes with the same name from libraries with different names (e.g. `counter` in Cyclone and Gem) is already a source of confusion for Pd users.

Suppose we were to release our work under a different name ("Recyclone," say), as a drop-in replacement for Cyclone. Recyclone would be identical to Cyclone in some respects, and its differences would be non-arbitrary because of the straightforward goal of Max/MSP compatibility. One obvious downside to this is that users who wanted to switch from Cyclone to Recyclone would need to change namespace settings in their existing patches. The larger point is that since the purpose of each branch differs only in that they target different versions of Max, they seem as much like predecessor and successor versions of the same software as the different versions of Max do. All of the predecessor's functionality is included in the successor's, with only minor changes.

The fact that two versions of the same library would be available via Pd's externals manager does not weaken our position. Indeed, multiple versions of many libraries are available, and it is up to the user to install whichever suits their needs. We believe that the disagreement is not about the simultaneous availability of two versions of the same library, but about the existence of two source branches with different maintainers/developers. This is more a social problem than a technical one. Provided that all parties are satisfied, we see no reason why cyclone-0.2 and cyclone-0.3 cannot exist side by side in the Pd ecosystem as a maintenance branch and development branch, respectively. While forking may be the normative strategy, forks make the most sense when each branch has ongoing diverging development.

Finally, we agree that Cyclone's system of modules is complicated; it is the primary motivation for reorganizing and streamlining the code. However, there is a great deal of useful code that need not be redesigned from scratch. We hope our efforts in increasing code transparency will encourage more participation from other developers.

Because many of the new Pd users in our community come from a Max/MSP background and tend to be daunted by Pd's steeper learning curve and relatively economical object collection, we believe that the project presented here will be an important first step toward preserving Cyclone's vital role in helping these users make that transition gracefully. However, we also believe that Cyclone is useful to Pd users in its own right. While we acknowledge that our development might overlap territory occupied by other libraries, we emphasize that Cyclone is a large library with many purposes outside of Max/MSP compatibility: it can replace many objects from unmaintained libraries, reduce the number of libraries a patch needs to load, or simply serve as a supplement to Pd vanilla. All of these purposes are significant justification for continual active development.

[14] These were all expressed in a discussion on The Pd-list from February 2016. See `https://lists.puredata.info/pipermail/pd-list/2016-02/113377.html`

5 Acknowledgements

There are many people we wish to thank. Krzysztof Czaja's initial effort was a large and worthy undertaking, and one of the most successful in the Pd world. Thanks to Hans-Christoph Steiner and Fred Jan Kraan for their stewardship; without them the library would likely be dead.

Katja Vetter's pd-lib-builder is an extraordinary tool for Pd developers, and has proved invaluable for understanding the structure of the Cyclone code and identifying opportunities for improvement.

We thank Ivica Ico Bukvic for the threaded version of `coll`, his ethic of active development, and his constant encouragement. Jonathan Wilkes's work on Purr Data will be very important to the future of Cyclone.

Thanks to Marco Matteo Markidis, Joel Matthys, and anyone else who has contributed code. Esteban Viveros and Flávio Schiavoni were instrumental in getting the process started on sure footing.

IOhannes m zmölnig merits gratitude for his leadership and advice. We thank David Zicarelli for his candid correspondence.

And of course we must thank Miller Puckette for inventing both Max and Pd.

References

[1] K. Czaja, "[PD] pd-max compatibility," April, 2002. `https://lists.puredata.info/pipermail/pd-list/2002-04/005949.html`

[2] K. Czaja, "[PD-announce] cyclone-0.1-alpha1," November, 2002. `https://lists.puredata.info/pipermail/pd-announce/2002-11/000139.html`

[3] H-C. Steiner, "[PD] Update cyclone maintenance," June, 2015. `https://lists.puredata.info/pipermail/pd-list/2015-06/110620.html`

[4] D. Zicarelli, "Max 5 and Attributes," October, 2007. `https://cycling74.com/2007/10/31/max-5-and-attributes/`