libpd: Past, Present, and Future of Embedding Pure Data

Peter Brinkmann
Google Inc
peter.brinkmann@gmail.com

Dan Wilcox
University of Denver
danomatika@gmail.com

Tal Kirshboim
tal.kirshboim@gmail.com

Richard Eakin
rich.eakin@gmail.com

Ryan Alexander
Okaynokay
scloopy@gmail.com

Abstract

libpd is a C library that turns Pure Data into an embeddable audio synthesis library, with wrappers for a range of languages and support for mobile platforms such as iOS and Android. We present an update on developments that have happened since PdCon11, including a discussion of libpd in creative environments and the emergence of apps that provide platforms for deploying Pd patches to mobile devices with little to no code. Looking forward, we discuss some challenges for future development, including questions of real-time safety and support for concurrency, as well as support for multiple instances. Finally, we examine the relationship of libpd to Pd itself.

Keywords

library, dsp, embedding Pure Data

1 Introduction

libpd began in July 2010, as a learning project whose only goal was to run Pd patches on Android devices. The core library emerged as a byproduct of a refactoring of the original solution. Within a few months of its publication, the team at RjDj decided to replace their in-house solution for embedding Pure Data with libpd. This led to the creation of Pd for iOS, whose open-source release, along with the creation of the Pd Anywhere forum at Create Digital Noise, marked the beginning of an exciting period of growth and creativity as libpd was adopted by hundreds of researchers and developers. Six years later, it is still going strong.

We presented the design of libpd and its language wrappers for Java and iOS at PdCon11 [1]. Much like Pd itself, the core library has remained rather stable, but thanks to the efforts of several dedicated contributors, the original, haphazard packaging has been replaced by proper release engineering. The integration of libpd into creative coding environments such as OpenFrameworks and Cinder as well as the emergence of general-purpose apps based on libpd have further supported the growth and adoption of libpd.

The purpose of this paper is to give an account of important technical developments around libpd that have occurred since PdCon11 and to identify problems that we will need to solve if libpd is to remain relevant, such as concurrency and support for multiple instances.

2 The Current State

Simply put, libpd *is* Pure Data vanilla without the graphical user interface or audio backends. It is not a fork of Pd. The Pure Data source code is included in the libpd git repository as a git submodule and directly tracks stable upstream releases. Further, any libpd-related bug fixes for the Pure Data core are submitted upstream to ensure the libpd codebase does not diverge. So far, this practice has propagated a number of fixes for 64bit related issues on iOS back into Pure Data with the help of Miller Puckette and community members.

2.1 Building

Building libpd is handled by a Makefile that compiles the core libpd C library as well as C++, Java, and C# wrappers. Make options also control the conditional compilation of libpd utilities and the bundled externals in the Pure Data `extra` directory. The Python wrapper is built using a traditional `setup.py` script that first builds the C library using the main Makefile. Obj-C support for iOS and macOS is provided via the included libpd Xcode project which is also utilized by the sample projects in the Pd for iOS repository.

2.2 Workflow and Release Cycle

The core libpd C API has been stable for a while. Most current work revolves around creation and maintenance of the various language wrappers. As of 2014, libpd uses semantic versioning and keeps a changelog between versions. Major changes are undertaken and tested in git branches, simple updates are committed to the master branch, which can be considered generally stable, and stable release versions are git tagged. Tagging allows for support by library management systems including CocoaPods via libpd's included libpd.podspec file and NuGet.

2.3 Documentation

High-level documentation comes in the form of the libpd website at http://libpd.cc, the libpd book "Making Musical Apps" [2], and sample projects such as those included with the Pd for iOS and Pd for Android repositories on GitHub. Low-level development documentation is available on the libpd GitHub wiki[1] which details build settings, per-language APIs, and working with libpd in various software environments. Supplementary information is largely community-based on the Pure Data mailing lists, forums, and via third parties through text and video tutorials.

2.4 C++ Wrapper

The development of the libpd C++ wrapper began in early 2011 as part of ofxPd, a libpd add-on library for the OpenFrameworks creative coding toolkit, and was integrated into libpd itself a year later. Influenced by the Java wrapper, the C++ interface is built around a class called PdBase which wraps the main libpd C API, base receiver classes for messages and MIDI data, and a set of convenience classes for the variable list data type and unique patch identifier. All classes are declared within the "`pd`" namespace. Additionally, PdBase provides a C++ style stream interface for message building and sending:

```
pd << StartMessage() << 1.23
   << "sent from a streamed list"
   << FinishList("fromCPP");
```

A minimal libpd C++ use case involves the following: implementing a subclass of PdReceiver and/or PdMidiReceiver for message handling; creating and initializing an instance of PdBase; creating a receiver subclass instance and setting PdBase to use it; subscribing to any required message sources; opening a patch; starting audio; and calling one of the PdBase process functions in an audio callback. The C++ wrapper does not contain an audio interface but can easily be dropped into a project using a cross-platform audio library such as PortAudio or RtAudio.

Optionally, PdBase can be built with a C++11 std::mutex to provide thread safety if the `LIBPD_USE_STD_MUTEX` compiler macro is defined, and it can be configured to pass messages through a lock-free ringbuffer in the C layer. Also, the C++ wrapper is forward-designed as PdBase currently utilizes a protected PdContext singleton class as a placeholder for possible upcoming multi-instance support.

2.5 ofxPd

ofxPd is a C++ add-on library for the OpenFrameworks creative coding toolkit that runs an instance of Pure Data within an OpenFrameworks application [3]. Audio, messages, and MIDI events can be passed to and from Pure Data patches with the OF run loop and the library is thread safe. The main ofxPd class is a convenience wrapper around the libpd C++ wrapper that adds support for multiple message and MIDI receivers, routing specific message sources to specific receiver class instances, and small changes such as updating MIDI channel ranges from 0-15 to 1-16 to match the numbers used in Pure Data itself.

Figure 1: NodeBeat scene on iPad

OpenFrameworks projects using ofxPd include NodeBeat, NinjaJamm, and Scrapple. Nodebeat is a node-based visual music app for iOS, Android, macOS, and Windows by Seth Sandler, Justin

[1] https://github.com/libpd/libpd/wiki

Windle, and Laurence Muller [4]. Record label Ninja Tune's NinjaJamm is a looper for Android and iOS with live effects and high quality, artist curated sample packs [5]. Scrapple is an interactive installation by media artist Golan Levin that uses computer vision to interpret the shapes of physical objects on a projected surface as a live visual score [6]. The audio engine for Scrapple was rebuilt in 2012 using ofxPd and features a bank of 128 FM synthesis voices.

2.6 Cinder and libpd

libpd can be used with the Cinder C++ creative coding library with an add-on called Cinder-PureDataNode [7]. Cinder has its own cross-platform, modular audio processing graph, in which the entire Pure Data context can be utilized as a node. The ability to process audio from both Cinder and Pd can be very powerful, allowing for the use of OpenGL for rich visuals, platform-specific file decoders and encoders, and low level DSP tools such as sample rate conversion.

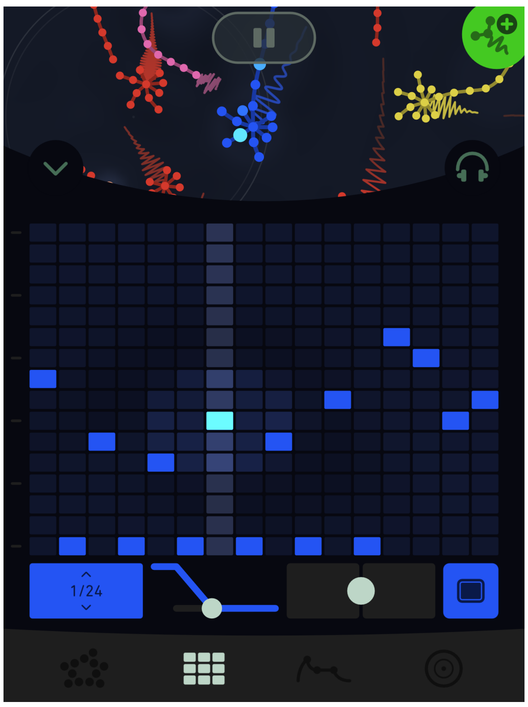

Figure 2: Seaquence on iPad

Cinder-PureDataNode was used extensively in the upcoming iOS app Seaquence [8], which features a stereo spacialized 10 voice, 50 note polyphonic subtractive synthesis engine built entirely with Pd. The ability to test and work on the synth outside the iOS app was a major win for Seaquence developers in terms of iteration time. Also, the ability to collaborate with knowledgeable members of the synthesis community was invaluable.

2.7 C# and NuGet

LibPDBinding, the C# wrapper for libpd was written by Tebjan Halm in 2012 for Windows .NET and acts as a thin layer around the C API with support for thread synchronization. In 2016, Thomas Mayer updated the build system to support 64bit on Windows and the MONO platform on Linux and macOS. Additionally, builds of libpd for C# are now hosted on NuGet [9], an open source package management system for Microsoft development, which greatly simplifies adding libpd to a C# project in Visual Studio or a supported MONO environment.

2.8 Pd for Android

Developing for the Android platform has changed substantially since the initial libpd release. Android Studio was released in late 2014 together with a new gradle build tool plugin for Android. These have replaced the Eclipse IDE and the ant build tool that were used for building Android applications until that time. The Pd for Android project was migrated to use this new toolchain in early 2015.

Another change that followed the migration to Android Studio was releasing Pd for Android as a maven artifact on JCenter [10]. Using JCenter to resolve Android dependencies is a common practice in Android development, and it significantly reduced the amount of effort involved in integrating Pd for Android in new Android apps. Prior to the release on JCenter, developers had to clone the project repository and its submodules and build the Java as well as the C code in the project, the latter using the Android NDK. With the JCenter dependency, all that is required in order to include Pd for Android in an app is to add a single line in the application's build file. When developers wish to use Pd externals which are not part of libpd in an Android app, they will still have to resort to the original and more complex way of integrating Pd for Android.

As part of the process of releasing Pd for Android on Jcenter, the btmidi[2] submodule that allows sending and receiving MIDI on Android devices was released on JCenter as well. While Android already includes MIDI support from version

[2]https://github.com/nettoyeurny/btmidi

6.0 Marshmallow on, the btmidi module allows devices with older Android versions to use MIDI.

Aside from the project-related structural changes mentioned here, Pd for Android proves itself to be very stable, even on newer Android versions. Problems that are reported with the library are often not specific to Pd for Android, but rather general issues with audio performance.

2.9 CocoaPods

As of 2013, libpd includes a podspec file for CocoaPods[3], a library dependency manager for iOS and macOS projects. This allows for the easy addition of libpd to an Xcode project without the need to manually include files and/or fiddle with build settings, thereby increasing deployment and accessibility. CocoaPods support was contributed to libpd by members of the CocoaPods community.

2.10 General Purpose Mobile Apps

Inspired by Reality Jockey's original RjDj app, a number of general purpose mobile applications for hosting libpd-based projects have been developed: DroidParty, MobMuPlat, and PdParty. Each of these apps can run Pure Data patches, access touch and sensor events, and support encapsulated scene directories including metadata, abstraction libraries, and sound files.

DroidParty by Chris McCormick pioneered the concept of recreating GUI objects (bang, toggle, hslider, etc) on Android [11]; Daniel Iglesia's MobMuPlat (Mobile Music Platform) for iOS and Android provides a custom GUI designer and supports OSC, MIDI, and networking features [12]; iOS app PdParty by Dan Wilcox faithfully replicates all Pure Data GUI objects, incorporates a web server and patch browser, and supports similar communication features to MobMuPlat [13].

3 Future Development

While the stability of libpd is generally a sign of success, we will need to make sure it remains relevant and maintainable as technology and engineering practices progress.

3.1 Visual Studio Support

Currently, the libpd C sources do not compile in Microsoft Visual Studio, mostly due to historical issues related to supported C versions that require changes to the Pure Data core. Consequently, builds of the library on Windows thus far have relied on MinGW. Newer versions of Visual Studio include C99, and it will be worthwhile to revisit any needed source updates to support building for platforms including Windows and Xbox, as well as the Windows app store.

3.2 Autotools

libpd currently relies on a single Makefile for most of its build and install options. As support for various languages, operating systems, and environments has grown, so has the complexity of the Makefile. A more sustainable long term approach for building, maintenance, and distribution would be to convert the project to use autoconf and automake which provide standard mechanisms for OS and compiler specific configuration, compile time options, and distribution tarball creation.

3.3 Alternatives to Locking

When processing audio in real time, it is generally advisable to avoid locks because of the risk of priority inversions. We do not agree with the frequently heard admonition to never use locks; since none of the popular operating systems are hard real-time systems, glitches are always possible, and so the question of whether to use locks is just another trade-off.

When working with a code base that predates ubiquitous multiprocessing, it is often hard or impossible to avoid locks. Pd falls into this category. libpd addresses this problem by deliberately discarding Pd's built-in global mutex lock, leaving the core library lock-free. That does not mean, however, that it is thread safe. Rather, the question of how to synchronize libpd calls is left to higher-level components, usually wrappers for languages like Java and C++.

The distribution of libpd includes a lock-free queue that is meant to reduce the need for locks. In practice, the trade-off we chose for most use cases is to pass outgoing messages (i.e. messages from Pd to the ambient app) through the lock-free queue, while passing incoming messages individually when holding a lock. The main reason for this is Pd's symbol table, which sits at the center of most of Pd's operations and is not thread safe.

While this approach has held up well in most cases, there is the possibility of audio dropouts

[3]https://cocoapods.org

when sending too many messages to Pd. Solving this in a general, performant way will probably require replacing Pd's symbol table with an implementation that is both thread and real-time safe.

3.4 Multiple Instances

The Pure Data core was originally written to be used within a single application and utilizes little data encapsulation, which makes it difficult to use within multi-context, multi-threaded environments such as reusable audio plugins. Preliminary work on multiple instance support by Miller Puckette in 2014[4] allows for context creation and switching with the `pdinstance_new()` and `pd_setinstance()` functions using a `t_pdinstance` type:

```
t_pdinstance *pd1 = pdinstance_new();
t_pdinstance *pd2 = pdinstance_new();
...
pd_setinstance(pd1); // 1st instance
libpd_openfile(argv[1], argv[2]);
pd_setinstance(pd2); // 2nd instance
libpd_openfile(argv[1], argv[2]);
```

This approach has been built upon by Kjetil Matheussen's Radium graphical DAW [14] and Pierre Guillot's Camomile VST plugin [15], both of which utilize custom wrappers of the Pure Data core with improved support for multiple instances and multithreading. Hopefully, this community work can be reincorporated into libpd and Pure Data.

3.5 libpd as a Core for Alternate GUIs

libpd's API is a standardized interface for message communication and sample I/O with the Pure Data DSP core. By abstracting much of the detail, omitting unused backends, and providing dummy interfaces, the changes made to the Pure Data source code allow for a simple embeddable C library suitable for use within other languages and environments beyond the desktop TCL/TK GUI.

Looking forward, this approach can be used to standardize communication between the core and the GUI, allowing for the creation of alternate editing GUIs using libpd. Jonathan Wilkes's GUI messaging specification for Purr Data provides a preliminary roadmap[5]. If this work can be integrated into the main Pure Data source and elements of the GUI code such as Undo/Redo be factored out into overridable C files with standard APIs, the various flavors of Pure Data (vanilla, Pd-L2ork, Purr Data) could all share the same upstream development source while maintaining their own customizations and improvements. Additionally, a future version of libpd with GUI messaging would facilitate the creation of even more esoteric editing environments such as an ncurses-based ASCII GUI or patch creation on mobile devices.

4 Conclusion

The first six years of libpd have seen the adoption and steady growth of a Pure Data-based ecosystem. Support for numerous programming languages and development environments has been added, leading to a new variety of audio visual apps, digital instruments, and sonic experiences. At this point, libpd development is largely stable but, moving forward, there are a number of improvements that can be made for libpd to work within audio plugins and as a core for new GUIs. The future is bright for Pure Data patches as the basis for continued computer music expression.

Acknowledgments

We thank everyone who contributed to libpd over the past six years, as well as everyone who made it a success by building amazing apps with it. It is an honor and a privilege to be a part of this community.

Peter Kirn maintains the libpd website: libpd.cc.

The development of ofxPd was supported by the CMU Frank-Ratchye Studio for Creative Inquiry and director Golan Levin.

References

[1] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner, "Embedding Pure Data with libpd," in *Pure Data Convention Weimar 2011*, 2011.

[2] P. Brinkmann, *Making Musical Apps: Real-time Audio Synthesis on Android and iOS.* O'Reilly Media, 2012.

[3] D. Wilcox, "ofxPd: a Pure Data addon for OpenFrameworks using libpd." [Online]. Avail-

[4] https://lists.puredata.info/pipermail/pd-dev/2014-05/019832.html
[5] https://git.purrdata.net/jwilkes/purr-data#gui-messaging-specification

able: https://github.com/danomatika/ofxPd

[4] "NodeBeat." [Online]. Available: http://nodebeat.com

[5] "NinjaJamm." [Online]. Available: http://www.ninjajamm.com

[6] G. Levin, "Scrapple." [Online]. Available: http://flong.com/projects/scrapple

[7] R. Eakin and R. Alexander, "Cinder-PureDataNode." [Online]. Available: https://github.com/notlion/Cinder-PureDataNode

[8] R. Alexander and G. Dunne, "Seaquence." [Online]. Available: http://okaynokay.xyz/presskit/seaquence

[9] "LibPDBinding on NuGet." [Online]. Available: https://www.nuget.org/packages/LibPdBinding

[10] "pd-for-android on JCenter." [Online]. Available: https://bintray.com/pd-for-android/maven/pd-for-android

[11] C. McCormick, "DroidParty." [Online]. Available: http://www.droidparty.net

[12] D. Iglesia, "MobMuPlat." [Online]. Available: http://danieliglesia.com/mobmuplat

[13] D. Wilcox, "PdParty." [Online]. Available: https://github.com/danomatika/PdParty

[14] K. Matheussen, Available: http://users.notam02.no/~kjetism/radium/

[15] P. Guillot, "Camomile." [Online]. Available: https://github.com/pierreguillot/Camomile/wiki