The Journal of Open Source Software

Liblsl.dart: A Dart native API for Lab Streaming Layer (LSL)

Luke Daniel Ring [1], **Anna Zamm** [1], **Chris Mathys** [2], **and Simon Lind Kappel** [3]

1 School of Communication and Culture, Department of Linguistics, Cognitive Science and Semiotics, Aarhus University, Denmark **2** School of Culture and Society - Interacting Minds Centre, Aarhus University, Denmark **3** Department of Electrical and Computer Engineering - Biomedical Engineering, Aarhus University, Denmark

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 10 October 2025
Published: unpublished

Summary

The `liblsl` Dart package is the first implementation of Lab Streaming Layer (LSL) in the Dart and Flutter ecosystem, enabling researchers to deploy multi-modal LSL-enabled data acquisition applications across mobile (iOS, Android) and desktop (Linux, macOS, Windows) platforms from shared source code.

Statement of Need

Neuroscience and behavioral research increasingly make use of consumer hardware due to its lower cost and sufficient performance for running experiments (Roque et al., 2025), however, the integration of consumer devices into laboratory data acquisition pipelines often requires platform-specific development and adds complexity to the data alignment and collection process, both of which can hinder flexibility and reproducibility (Iwama et al., 2024). The LSL system partially addresses some of the complexity via a software based unified method of time-synchronized data acquisition across heterogeneous hardware and software systems (Kothe et al., 2025). LSL handles clock synchronization, network communication, and data buffering, making it particularly valuable for applications requiring precise temporal alignment of multiple data sources. The `liblsl` Dart package further reduces the complexity by enabling researchers to deploy LSL-enabled applications across all major platforms from a single Dart/Flutter codebase. Potential use cases include general multimodal experiments (Dolmans et al., 2020; Wang et al., 2023), mobile brain-computer interfaces using an EEG headset and a smartphone (Blum et al., 2017; Debener et al., 2015; Stopczynski et al., 2014), hyperscanning studies (Zamm et al., 2024) that simultaneously collect neuroimaging data from multiple participants (Boggio et al., 2023; Luft et al., 2022; Roque et al., 2025), and distributed experiments where multiple labs using a variety of devices collect methodologically consistent data in a standardized format for analysis (Demazure et al., 2021; Schultz et al., 2021).

State of the Field

There are existing LSL implementations in other programming languages that are platform-specific or restricted to desktop devices (Kothe et al., 2019), this package leverages Dart/Flutter's cross-platform capabilities while maintaining the millisecond level latency requirements of neurophysiological research through direct Foreign Function Interface (FFI) bindings to the native LSL C library (Stenner et al., 2023). The need for a Dart LSL package is particularly extant within the behavioural science research community, where researchers

often require multiplatform applications but lack the ability to develop applications for new platforms.

Software design

Due to `liblsl.dart` being in a library implementation, the intentional decision was made to create a simplified wrapper in addition to exposing direct access to the `liblsl` API via the Foreign Function Interface (FFI) generated by `ffigen`[1]. The native interface requires knowledge of low-level memory management and a familiarity with the FFI API, whereas the wrapped API handles the lower-level complexity and provides clear, typed Dart objects for working with LSL. `liblsl.dart` allows for both pseudo-synchronous and asynchronous, multithreaded (`Isolates` in Dart) interaction with LSL, giving users the ability to decide for their use-case if stricter timing or increased data throughput are more relevant. Although the current underlying `liblsl` implementation is asynchronous, the synchronous version in Dart removes the overhead of inter-thread communication and provides the foundation for integrating an upcoming `sync` mode in the `liblsl` library. The asynchronous version in Dart prevents blocking during high-frequency polling, which is especially relevant in Flutter where the code can be executed in the UI thread where blocking will make an application unresponsive to end users. From the start, this package was required to maintain the low latency of `liblsl` to ensure it is appropriate for online multimodal research. Comprehensive testing of the package is done via unit testing and continuous integration via GitHub Actions.

Performance

Performance was characterized under controlled conditions to isolate different sources of latency. Local device tests measured the computational overhead of the package's Dart API by having a single device both produce and consume 1000 Hz data streams (16 channels, float32 format), representing typical EEG recording parameters. Network tests used two iPad Pro M4s connected via 1Gbps USB-C Ethernet adaptors to a consumer-grade gigabit router. All tests were run for 3 minutes (approximately 180,000 samples) to ensure statistical reliability. Measurements represent end-to-end latency from sample production to consumption, including API call overhead, serialization, network transmission, and deserialization.

[1] ffigen: bindings generator for FFI bindings. https://pub.dev/packages/ffigen.

The Journal of Open Source Software

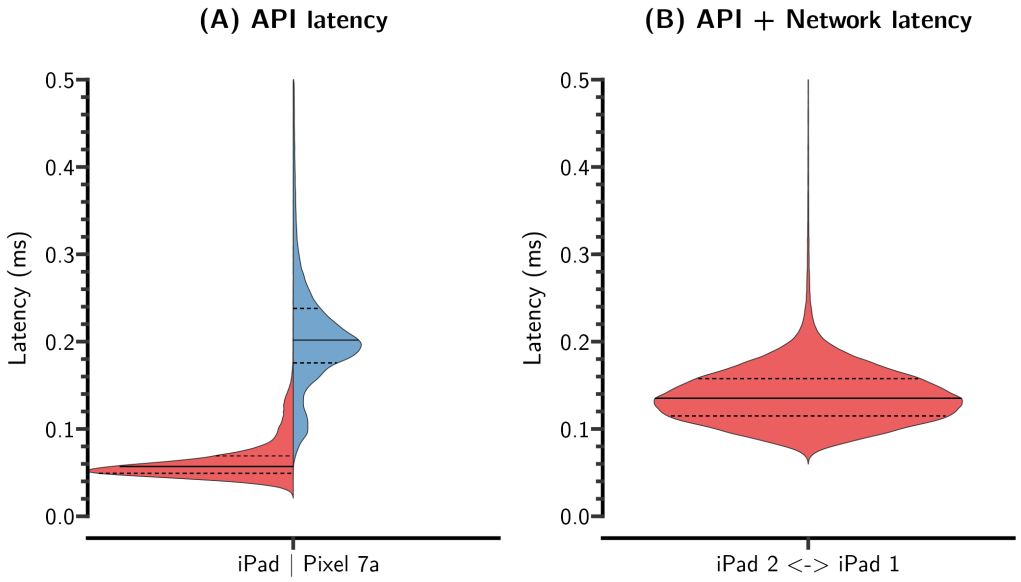

Figure 1: Latency characterization of the `liblsl` Dart package. **(A)** Local performance on iPad Pro M4 (left) and Pixel 7a (right), each producing and consuming its own 1000 Hz data stream. **(B)** Network performance between two iPads over 1Gbps Ethernet. Violin plots show median (solid line) with first and third quartiles (dashed lines). See Table 1 for complete statistics. *Note: Outliers >500µs were excluded from visualization but are included in statistics.*

Table 1: Summary of latency measurements.

Condition	Device	n	Min (µs)	Max (µs)	Mean (µs)	SD (µs)
Local	iPad Pro M4	180000	21	4264	65	47
Local	Pixel 7a	179900	42	18809	274	434
Network	iPad Pro M4	180000	60	4261	148	129

Table 1 summarizes latency statistics across conditions and devices. Figure 1 illustrates the distribution of the relevant performance characteristics: (A) single-device performance showing the API's computational overhead when producing and consuming data locally (iPad: µ=65µs, σ=47µs; Pixel 7a: µ=274µs, σ=434µs), where standard deviations reflect timing jitter inherent to the operating system's thread scheduling and (B) network performance between two iPads over a 1Gbps wired connection (µ=148µs, σ=129µs). These results confirm that the Dart wrapper introduces minimal overhead beyond the base LSL C library performance, with observed differences primarily attributable to device hardware capabilities and network infrastructure rather than the API implementation itself.

The observed latencies are consistent with previous benchmarking of native LSL implementations (Kothe et al., 2025) and demonstrate that this package preserves the real-time performance characteristics required for neurophysiological applications (Iwama et al., 2024). The local processing latencies (65-274µs) are well below typical neurophysiological event timing requirements (<1ms), confirming suitability for EEG, electromyography (EMG), and other biosignal applications[2]. The mean inter-device network latency (148µs) demonstrates that network overhead remains minimal on well-configured local networks[3], though researchers

[2]Raw data and source code used for generating the figures and statistics are available at: https://github.com/NexusDynamic/liblsl.dart/tree/main/packages/liblsl/paper/analysis.

[3]In a lab context, this typically may be achieved by using a closed, wired network, with a high throughput hardware switch or router, and by disabling firewalls, traffic shaping and other features that may introduce latency such as deep packet inspection.

should note that wireless networks and other factors including network traffic congestion may introduce additional jitter. The device-dependent variation (iPad: 65µs vs Pixel 7a: 274µs) reflects differences in CPU architecture, device network hardware and operating system scheduling rather than API limitations, indicating that platform selection should be validated through careful testing based on the requirements of the specific application.

Research Impact Statement

LSL lowers the complexity of multimodal time-synchronized data acquisition from multiple devices (Dolmans et al., 2020; Iwama et al., 2024; Kothe et al., 2025), and Dart/Flutter simplify cross-platform application development. Together, they provide a powerful toolset for researchers and developers to create flexible, reproducible, and accessible applications for data collection and analysis. The package includes well-documented source code, examples, tests and related packages that implement `liblsl.dart`. During the development of this package, the author has contributed fixes and improvements to `liblsl`[4] which benefits this package as well as the core library and implementations in other programming languages. From a practical standpoint, the `liblsl.dart` package has also been used to build the LSL library for previously unsupported platforms[5] which demonstrates the flexibility of the current implementation (see also Reference Implementations).

Example LSL Integration

The following code demonstrates how a complete data streaming application can be built in under 30 lines of code:

```dart
import 'package:liblsl/lsl.dart';
import 'dart:async';

void main() async {
  // Describe the stream
  final info = await LSL.createStreamInfo(
    streamName: 'Counter',
    streamType: LSLContentType.markers,
    channelCount: 1,
    sampleRate: LSL_IRREGULAR_RATE,
    channelFormat: LSLChannelFormat.int8,
    sourceId: 'uniqueStreamId123',
  );
  // Create the outlet
  final outlet = await LSL.createOutlet(streamInfo: info);

  // Send data at 2 Hz (configurable for your application)
  for (var i = 0; i < 10; i++) {
    final sample = [i];
    outlet.pushSample(sample);
    await Future.delayed(const Duration(milliseconds: 500));
  }
  // Clean up
  outlet.destroy();
  info.destroy();
}
```

[4] Merged and pending pull requests by author Luke Ring (@zeyus) for the `liblsl` repository: https://github.com/sccn/liblsl/pulls?q=is%3Apr+author%3Azeyus.

[5] `liblsl` for ARMv7 on Debian 9 built with `liblsl.dart`: https://github.com/NexusDynamic/Bela-liblsl.

Reference Implementations

The `liblsl.dart` package is currently being used actively in research projects by the authors which will be published as partial requirements of a PhD thesis. The LSL integration is used in a multimodal tablet-based experimental platform built in Flutter designed for simultaneous data acquisition from groups of participants interacting in real-time[6]. The platform has also been utilized for a multi-device coordination system that can control network device configuration and data stream management in lab settings[7]. This package was also used to create a latency and timing benchmarking application that validates the configuration in lab-specific environments[8].

AI Usage Disclosure

The package and wrapping API design are that of the author's and are the result of incremental development and testing. However, two generative AI tools have been used during development: 1) Microsoft Copilot integration in Visual Studio Code has been used as an enhanced auto-correct tool for simple or repetitive tasks (i.e. class structure skeletons, paths and import statements); 2) Claude Code has been used for "rubber duck debugging", general debugging, and for limited generation of boilerplate code, some tests and documentation. No AI generated content has been accepted without review and testing, nor does it constitute the majority of the work produced. In addition to human review, the suite of tests, including performance testing are continuously run to prevent regressions. For this paper, Claude was used to suggest edits for clarity and structure, but was not used to generate the contents or sections, nor were the suggestions implemented wholesale.

Acknowledgements

This library builds on `liblsl` by Christian A. Kothe using the Dart programming language by the Dart project authors, Google. Thanks to Chadwick Boulay and members of the dart_community Discord for help with debugging. No external funding was used for the development of this package and is being developed as part of the first author's PhD research project, which was awarded a "4+4 scheme" research scholarship from the Arts Graduate School at Aarhus University.

References

Blum, S., Debener, S., Emkes, R., Volkening, N., Fudickar, S., & Bleichner, M. G. (2017). EEG Recording and Online Signal Processing on Android: A Multiapp Framework for Brain-Computer Interfaces on Smartphone. *BioMed Research International*, *2017*(1), 3072870. https://doi.org/10.1155/2017/3072870

Boggio, P. S., Wingenbach, T. S. H., Da Silveira Coêlho, M. L., Comfort, W. E., Murrins Marques, L., & Alves, M. V. C. (Eds.). (2023). *Social and Affective Neuroscience of Everyday Human Interaction: From Theory to Methodology*. Springer International Publishing. https://doi.org/10.1007/978-3-031-08651-9

Debener, S., Emkes, R., De Vos, M., & Bleichner, M. (2015). Unobtrusive ambulatory EEG using a smartphone and flexible printed electrodes around the ear. *Scientific Reports*, *5*(1), 16743. https://doi.org/10.1038/srep16743

[6] Poster overview: https://nexusdynamic.org/FINAL-Coop_comp_paradigm-A0Poster_reduced.pdf.

[7] `liblsl_coordinator`: https://github.com/NexusDynamic/liblsl.dart/tree/main/packages/liblsl_coordinator.

[8] `liblsl_timing`: https://github.com/NexusDynamic/liblsl.dart/tree/main/packages/liblsl_timing.

Demazure, T., Karran, A. J., Boasen, J., Léger, P.-M., & Sénécal, S. (2021). Distributed Remote EEG Data Collection for NeuroIS Research: A Methodological Framework. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Augmented Cognition* (pp. 3–22). Springer International Publishing. https://doi.org/10.1007/978-3-030-78114-9_1

Dolmans, T., Poel, M., van't Klooster, J., & Veldkamp, B. (2020). Data synchronisation and processing in multimodal research. *Measuring Behavior 2020-21 Volume*, *1*, 26–32.

Iwama, S., Takemi, M., Eguchi, R., Hirose, R., Morishige, M., & Ushiba, J. (2024). Two common issues in synchronized multimodal recordings with EEG: Jitter and latency. *Neuroscience Research*, *203*, 1–7. https://doi.org/10.1016/j.neures.2023.12.003

Kothe, C., Medine, D., Boulay, C., Grivich, M., & Stenner, T. (2019). *Liblsl language wrappers — Labstreaminglayer 1.13 documentation*. https://labstreaminglayer.readthedocs.io/info/language_w

Kothe, C., Shirazi, S. Y., Stenner, T., Medine, D., Boulay, C., Grivich, M. I., Artoni, F., Mullen, T., Delorme, A., & Makeig, S. (2025). The lab streaming layer for synchronized multimodal recording. *Imaging Neuroscience*, *3*, IMAG.a.136. https://doi.org/10.1162/IMAG.a.136

Luft, C. D. B., Zioga, I., Giannopoulos, A., Di Bona, G., Binetti, N., Civilini, A., Latora, V., & Mareschal, I. (2022). Social synchronization of brain activity increases during eye-contact. *Communications Biology*, *5*(1), 412. https://doi.org/10.1038/s42003-022-03352-6

Roque, T. R., Sun, R., Do, Y. L. E., Maldonado, D. L., & Leslie, G. (2025). Real-Time Mobile EEG Hyperscanning: A Precise and Accessible Platform for Social Brain–Computer Interfaces. *IEEE Sensors Journal*, 1–1. https://doi.org/10.1109/JSEN.2025.3597568

Schultz, T., Putze, F., Fehr, T., Meier, M., Mason, C., Ahrens, F., & Herrmann, M. (2021). *Linking Labs: Interconnecting Experimental Environments* (No. arXiv:2102.03684). arXiv. https://doi.org/10.48550/arXiv.2102.03684

Stenner, T., Boulay, C., Grivich, M., Medine, D., Kothe, C., Tobiasherzke, Chausner, Grimm, G., Xloem, Biancarelli, A., Mansencal, B., Maanen, P., Frey, J., Jidong Chen, Kyucrane, Powell, S., Clisson, P., & Phfix. (2023). *Sccn/liblsl: V1.16.2*. Zenodo. https://doi.org/10.5281/ZENODO.5415958

Stopczynski, A., Stahlhut, C., Larsen, J. E., Petersen, M. K., & Hansen, L. K. (2014). The Smartphone Brain Scanner: A Portable Real-Time Neuroimaging System. *PLOS ONE*, *9*(2), e86733. https://doi.org/10.1371/journal.pone.0086733

Wang, Q., Zhang, Q., Sun, W., Boulay, C., Kim, K., & Barmaki, R. L. (2023). A scoping review of the use of lab streaming layer framework in virtual and augmented reality research. *Virtual Reality*, *27*(3), 2195–2210. https://doi.org/10.1007/s10055-023-00799-8

Zamm, A., Loehr, J. D., Vesper, C., Konvalinka, I., Kappel, S. L., Heggli, O. A., Vuust, P., & Keller, P. E. (2024). A practical guide to EEG hyperscanning in joint action research: From motivation to implementation. *Social Cognitive and Affective Neuroscience*, *19*(1), nsae026. https://doi.org/10.1093/scan/nsae026