# PSST: Enabling Blind or Visually Impaired Developers to Author Sonifications of Streaming Sensor Data

Venkatesh Potluri
University of Washington
Seattle, WA, USA
vpotluri@uw.edu

John R. Thompson
Microsoft Research
Redmond, WA, USA
johnthompson@microsoft.com

James Devine
Microsoft Research
Cambridge, United Kingdom
james.devine@microsoft.com

Bongshin Lee
Microsoft Research
Redmond, WA, USA
bongshin@microsoft.com

Nora Morsi
University of Washington
Seattle, WA, USA
morsin@cs.washington.edu

Peli de Halleux
Microsoft Research
Redmond, WA, USA
jhalleux@microsoft.com

Steve Hodges
Microsoft Research
Cambridge, United Kingdom
steve.hodges@microsoft.com

Jennifer Mankoff
University of Washington
Seattle, WA, USA
jmankoff@cs.washington.edu

## ABSTRACT

We present the first toolkit that equips blind and visually impaired (BVI) developers with the tools to create accessible data displays. Called PSST (Physical computing Streaming Sensor data Toolkit), it enables BVI developers to understand the data generated by sensors from a mouse to a micro:bit physical computing platform. By assuming visual abilities, earlier efforts to make physical computing accessible fail to address the need for BVI developers to access sensor data. PSST enables BVI developers to understand real-time, real-world sensor data by providing control over what should be displayed, as well as when to display and how to display sensor data. PSST supports filtering based on raw or calculated values, highlighting, and transformation of data. Output formats include tonal sonification, nonspeech audio files, speech, and SVGs for laser cutting. We validate PSST through a series of demonstrations and a user study with BVI developers.

## CCS CONCEPTS

• **Human-centered computing** → *User interface toolkits*; **Accessibility technologies**.

## KEYWORDS

Accessibility, Physical Computing, Blind or Visually Impaired (BVI) Programmers, Toolkit

**ACM Reference Format:**
Venkatesh Potluri, John R. Thompson, James Devine, Bongshin Lee, Nora Morsi, Peli de Halleux, Steve Hodges, and Jennifer Mankoff. 2022. PSST: Enabling Blind or Visually Impaired Developers to Author Sonifications of Streaming Sensor Data. In *The 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22), October 29-November 2, 2022, Bend, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3526113.3545700



*UIST '22, October 29-November 2, 2022, Bend, OR, USA*

ACM ISBN 978-1-4503-9320-1/22/10.
https://doi.org/10.1145/3526113.3545700

## 1 INTRODUCTION

The Blind and Visually Impaired (BVI) community has a long history of expertise and interest in physical computing, starting with the publication of the *Smith-Ketterwell Technical File* [26] in 1980. This publication, edited by a blind engineer, demonstrated that BVI people can accessibly make useful physical computing devices that creatively address a variety of needs. The field of physical computing has evolved dynamically and rapidly [40, 41] since the last issue of the *Technical File* was published in 1998. Today's systems are far easier to learn, have demonstrated value as a training ground for future STEM workers [15, 23], and can also help to diversify STEM (*e.g.,* [11, 30, 31]). However, these advances have often not been accessible. Recently, attention has turned to whether people with disabilities are included by these technologies [69]. Although maker spaces have become a home for the do-it-yourself assistive technology (DIY-AT) movement [12, 13, 34, 48], these movements are sometimes structured as "for" people with disabilities rather than "with" or "by" them.

BVI people's interest in physical computing has evolved with the field. For example, the Blind Arduino Blog (blarbl.blogspot.com) contains advice and tutorials on how to build circuits using the Arduino. Recent work has begun to raise and address this inaccessibility and exclusion through participatory design sessions and digital tools [18, 33, 58, 66, 69]. Other research has included workshops on nonvisual soldering, tactile schematics, and Arduino programming [60–62].

While this body of work is essential, its success only makes access to data more urgent. Regardless of visual capacity and accessibility of maker spaces, physical computing can be challenging to learn. System state is often not visible [47] and beginners also need to develop basic understanding of sensors [15]. As a result, many

bugs are related to a lack of understanding of sensors and unexpected sensor readings [8]. Creating physical computing solutions requires understanding, configuring, and potentially debugging sensors that provide information about the physical world, such as accelerometers and light sensors. This in turn requires the ability to understand the data these sensors generate.

Visualizations of sensor state are available in many of today's physical computing platforms in visual form (*e.g.,* [27, 68]). However, most of the websites and tools developed to program these boards, as well as the tutorials created for them, are partly or wholly inaccessible. For example, in their study of over 3,000 online tutorials, Davis *et al.* [16] found that less than 2% met web accessibility criteria. A small body of work has begun to address accessibility of physical computing [7, 36, 52]. However, data visualization has not been a focus of these works.

Additionally, most tools to program accessibly [5, 59, 65] do not explore accessibility of sensor data. Recent works have explored data sonification outside of the programming domain [38]; however, they have almost entirely focused on predefined visualizations of *static* data. Understanding sensor data requires observing and interacting with dynamic visualizations of streaming data.

We argue that an accessible tool for understanding sensors must support streaming data and enable BVI people to author their own visualizations to access and interpret data generated by sensors. We combine real-time data sonification techniques and screen reader interactions to build these auditory displays and provide a tool for BVI developers to build their own auditory displays. Through the design, development, and evaluation of our Physical computing Streaming Sensor data Toolkit (PSST), we contribute:

- A set of design requirements that describes the interaction needs for understanding sensor data, including response speed, calibration, and noisiness. We map these requirements onto specific interaction techniques, such as setting an alert to monitor when a continuous, linear data stream reaches a certain range; speaking out the value of a data point whenever the slope changes (*e.g.,* when a wave peaks); or mapping each incoming point to the audible range.
- A data display toolkit, the Physical computing Streaming Sensor data Toolkit (PSST), for researchers, including those who are BVI, to explore designs of accessible data displays. PSST provides BVI developers with the tools to build customized data displays using feedback mechanisms including spoken text, stereo control over sonification and prerecorded sounds, and physical artefacts created with a laser cutter. In addition, PSST provides a general structure for sonifying information computed over the data, which allows for actions such as alerting the user to specific values or filtering data. Finally, PSST can easily be extended with new outputs and computations as a BVI developer feels necessary.
- The PSST dashboard, an accessible data dashboard that provides a multi-modal interface to use our toolkit. While the toolkit can be used through code, our dashboard allows users to customize displays using an accessible graphical user interface. We have connected the dashboard to the micro:bit [4] and a plug-and-play physical computing platform, called Jacdac [50], allowing end users who are new to physical computing to easily use PSST without programming.

We validate the power and coverage of our toolkit through a series of demonstrations. First, we show its flexibility with respect to input by applying it to static data, mouse and keyboard data (including creating a piano), and streaming sensor data. Second, we show its flexibility with respect to output by using a variety of audio-based outputs and additionally creating a laser cut data log that is not only tangible but can also be used with a standard punch-programmed music box to create hand-cranked sonification. Finally, through a study with two BVI developers using the PSST dashboard to perform tasks that map to our design goals, we validate the toolkit's ability to enable BVI developers to understand sensor data.

Although PSST is a powerful and general tool for sonifying any streaming data, our primary motivation is to solve *in-situ* accessibility challenges that prevent BVI developers from understanding sensor data, an area we believe has been overlooked by accessible programming tool researchers. In the sections that follow we focus on this primary goal.

First, we review literature relevant to accessible understanding of sensor data, deriving a set of design goals for understanding sensor data. Next, we introduce PSST using a scenario, and then describe its architecture, library, and Graphical User Interface (GUI). Following this, we provide some demonstrations of how PSST can be used for a variety of input and output scenarios. We conclude with a study with two BVI participants that demonstrates the power of the PSST GUI for solving tasks that map onto all of the design goals synthesized from our literature survey.

## 2 BACKGROUND AND REQUIREMENTS

Prior work has looked at several aspects of physical computing accessibility for BVI people, including tooling for understanding hardware circuits [26], such as oscilloscopes and continuity testers, tactile circuit diagrams [16, 60], and soldering [62]. Traditionally these have been necessary preconditions for working with physical circuits. However, the advent of recent physical computing boards, such as the micro:bit [4], has obviated the need for some of these low-level tools while introducing largely inaccessible new block-based programming languages [44, 51] and web-based tutorials [16]. While alternatives to block-based languages are available on a range of platforms from the Arduino to the micro:bit, accessible data visualization for understanding sensors is not well supported.

More generally, visualization consumption and authoring are both under-supported for BVI people. In Kim *et al.'s* survey of 56 accessible visualization papers published between 1999 and 2020, the majority were intended for use in static contexts [38]. Eleven supported the creation of a chart that is accessible to BVI users but only two papers supported interactive authoring of charts by BVI users. Interaction with streaming data was not mentioned at all. Even basic information such as discoverability and comprehensibility is not supported by many online visualizations [67]. However, when sonification is available, its impact is powerful, as demonstrated by the SonicX system [24], designed for collaborative

use by sighted and Blind astronomers [49]. The system led to new scientific discoveries.[1]

Further, streaming data has special requirements that may not be well-supported by most existing approaches to accessible visualization. In Kim *et al.'s* survey of accessible visualization research, 31 of the 56 papers used multiple modalities [38], with some combination of Braille, haptic or tactile output, combined with some kind of audio output (speech, sonification, or both) being by far the most common approach. However, the most commonly used physical outputs—embossed prints or other physical visualizations (19 of 56)—were only appropriate for static, not streaming data. Haptics can be used for physical output, but again are not naturally suited to streaming data, which is constantly changing and thus hard to explore in this way. Similarly, speech may be ill-suited for expressing streaming data due to the difficulty of summarizing dynamically-changing data verbally.

To conclude, an understanding of the requirements for accessible interactive visualization of sensor data is not directly available in the literature. To address this, we first survey the types of capabilities found in streaming data visualization tools, and then highlight requirements for data exploration and accessibility opportunities associated with these tools. Next we survey tools supporting development and debugging of physical and embedded computing programs, highlighting their consistent use of visualization and their lack of accessibility. Finally, we bring these together into a set of requirements for a tool that allows the authoring of streaming sensor data visualizations.

## 2.1 Sonification of Streaming Data

Sonification has been identified as an important technique for monitoring streaming data in various domains including finance [37], programming [35], business process monitoring [29], system administration [63], human activity data visualization [14], and accessibility [53]. Some important facets of monitoring identified in the literature include alerting and understanding both individual streams of data and the overall data landscape [63]. Sonification also has a long history in both art and science (*e.g.,* [20, 72]).

A variety of techniques for sonification have been studied over many decades, ranging from orchestration to spatialization [63]. The field is sufficiently mature that sonification books (*e.g.,* [28]) and design guides (*e.g.,* [17, 20, 21, 54]) have been published. Several authors have suggested basic capabilities a data exploration system should support, whether sonification-based or not. As summarized by Beilharz and Ferguson [6], these include the ability to parse, filter, mine, represent, refine, and interact with data [22], as well as the ability to analyze trends, detect patterns, and estimate and compare points [75]. To support these capabilities, it may be helpful to indicate dispersion, slope, and/or various types of statistical summaries, including means, medians, and modes. Beilharz and Ferguson [6] go on to suggest that sonification displays can support these goals by assigning timbre, pitch, duration, or intensity to data points that share certain characteristics, and argue for minimalism in audio design. At the same time, the ability to "foreground critical data-moments in relation to thresholds and constraints set by the user…is an important aspect of the customizable interface and interactivity, allowing the user to define importance and re-examine the dataset according to different measures." [6]. They also discuss interactivity in the context of a static dataset, highlighting the ability to explore the sonic space in 2D and 3D using appropriate controllers. Some interactive techniques such as interactive control over scaling and filtering might translate well into the streaming context. Finally, Beilharz and Ferguson [6] discuss methods for handling multiple concurrent information streams, such as using different "pitches register, timbres and filtering to distinctively characterise competing elements, as well as temporal off-set (rhythmic separation, asynchronicity)."

These methods have in turn been codified in sonification toolkits (*e.g.,* [42, 43, 46]) and sonification authoring languages [3, 43, 57]. Common capabilities provided by these tools include analyzing and transforming data including rescaling and quantization, filtering, inversion, and normalization. A second set of capabilities involves mapping this transformed data onto sonification parameters, including rhythm, articulation, pitch, and timbre. Often these are embedded in Turing-complete visual or textual languages, opening the door to an endless range of data transformation and mapping possibilities. However, these tools are not designed for BVI people, and often emphasize ease of use by (seeing) non-programmers. Both of these trends mean that a number of sonification toolkits are structured around inaccessible audio-visual interactions either at the time when sonifications are designed, when they are used, or both (*e.g.,* [14, 43, 57]). For example, Rotator [14] expects sonifications to be configured in a GUI and employs D3 visualizations, which are generally not accessible [67]. Similarly, AeSon [6] provides both a visual language and a GUI for authoring sonifications and SIFT [10] is a block-based language (such languages are generally not designed for accessible to BVI people [44]).

A subset of these tools are programming-based, allowing the authoring of sonifications without being GUI dependent. DTM [73, 74] supports JavaScript-based code specification for data sonification. It can analyze and transform arbitrary data sources, and then map them to sound using the Web Audio API. Because of its programmable nature, it is an easily extensible tool with libraries for statistics calculation and data manipulation, as well as a sonification library that includes "rhythmicization, note dynamics, articulation, pitch modulation, pitch scale, chord voicing, timbre modulation, ... [and common] musical patterns." Rescaling, quantization, and so on are easy to support in a programmatic tool like this. The system makes use of JavaScript function chaining for consecutive application of toolkit elements, and is stream-based. SoniPy [76] similarly supports a range of data filtering and modification techniques, in combination with sonification. This ability to merge data processing and sonification is a powerful tool in a programming environment. SonicPi [1] also supports music authoring through code by providing a domain-specific language that simplifies authoring while being fully featured enough to be used for learning to program.

However, these programming tools, while extremely powerful, do not incorporate the full range of sonification capabilities that a person focused primarily on nonvisual access to data might care to include. For example, multimodal output is often not supported (such as sonification plus speech or the use of easily recognizable short sound clips). Further, while filtering and scaling are supported,

[1] TED talk: How a blind astronomer found a way to hear the stars

| Design Goal | Realization in PSST |
|---|---|
| **DG1** Learn how sensors respond to stimuli [15] | Provide real-time access to streaming sensor state |
| **DG2** Identify appropriate thresholds for sensors [8] | Provide reactive filters that highlight extremes as the user explores a sensor's range |
| **DG3** Learn about unexpected sensor readings [8] | Let the user highlight sudden changes or unusual values |
| **DG4** Analyze trends [75] and provide statistical summaries [6] | Support transformations and calculations of statistics over a stream, such as sonifying the slope, or calculating a running average |
| **DG5** Display multiple concurrent information streams [6] | Select and assign sonifications to any number of different sensor streams |

**Table 1: A list of capabilities derived from our literature survey and the features of PSST that support them. Overall, PSST has the goal of making sensor state "visible" [47].**

the programming abstractions provided are not specific to streaming sensor data. While these languages are sufficiently general to support anything in principle, good abstractions for filtering, highlighting, and analyzing streaming data are important to lower the floor for entry by beginners.

## 2.2 Capabilities Relevant to Sensor Data

To understand what abstractions are relevant to analyzing streaming sensor data, we turn to literature on understanding sensor data and physical computing.

In a study comparing the efficacy of physical computing to other methods for teaching programming, Chung *et al.* [15] found that some of the learning goals students needed to achieve included basic understanding of how and why sensors responded to stimuli, and analog to digital conversion. Another study by Booth *et al.* [8] found problems such as using the wrong thresholds for a sensor, in addition to multiple hardware bugs that were visible in unpredictable sensor readings. Understanding state is a general problem that is critical in all programming tasks [39]. It is, however, particularly hard to visualize the state in physical computing and embedded systems because the state can change so dynamically, and may not be stored in a variable.

In response to the need to understand and debug physical computing and embedded systems, a variety of tools and research projects have incorporated support for data visualization. The Arduino has the ability to graph output sent to its serial port [78] and the micro:bit's Jacdac programming interface includes visualizations showing the current status of all sensors [4]. Similarly, the Data Sensor Hub(DaSH) [25] is a tool built with the micro:bit that enables students to measure and analyze data collected from a variety of sensors. Several systems show the voltage or current flowing through a circuit or breadboard [19, 55, 77]. Scanalog visualizes analog circuits and interactive tuning of signal transformations [71]. Bifröst went beyond visualization to lay out five design goals: (1) help users localize whether a fault is in hardware or software; (2) make internal state visible; (3) provide context to help users interpret visible behaviors; (4) automate testing and hypothesis generation; and (5) link views (traces and code) [47]. In a study, Bifröst successfully supported both diagnostic and debugging tasks.

Two things are important to note about these tools. Firstly, to the best of our knowledge, they are not accessible to BVI people. The research papers do not mention accessibility or inclusion of BVI participants; the Jacdac website does not use accessible visualizations, and the Arduino visualization is also not accessible (though it is possible to save serial output to a file and then explore it). Secondly, a common theme to all of these tools is that live visualization of data about system and/or sensor state is of value in supporting their target population.

## 2.3 Requirements Summary

To summarize, neither sonification toolkits, nor tools for learning about and debugging physical and embedded computing, currently provide an accessible solution for BVI people to understand the state of sensors or easily explore streaming data. However, an extensive body of work in these domains gives us clear guidance about requirements for such systems. We have synthesized these into five primary goals, listed in terms of the end user's goals, in Table 1. They are (1) Learn how sensors respond to stimuli; (2) Identify appropriate thresholds for sensors; (3) Learn about unexpected sensor readings; (4) Analyze trends and provide summaries; and (5) Do this for multiple concurrent information streams. Each of these five goals is supported by PSST, as we summarize in Table 1 and describe in more detail in the next section.

# 3 PHYSICAL COMPUTING STREAMING SENSOR DATA TOOLKIT (PSST)

The Physical computing Streaming Sensor data Toolkit (PSST) is designed to simplify the authoring of sonification of live sensor data for the purposes of understanding, testing, and debugging sensors. It is designed to support authoring visualizations, typically sonifications, of different types of streaming data, while prioritizing features that will be helpful when dealing with multiple sensor data streams. To author a visualization, the toolkit users, such as BVI programmers, will use combinations of *Data Sinks*, *Data Handlers*, and *Datum Outputs*, all controlled by a single *Output Engine*.

As shown in Figure 1, sonification starts with a class called `OutputEngine`, which is the repository for all the streams that are being sonified. Each stream is associated with a `DataSink`, which keeps track of the processing pipeline for the incoming data (the list or lists of `DataHandlers` and their associated `DataOutputs`). `DataSinks` can receive incoming streaming data from any RxJs

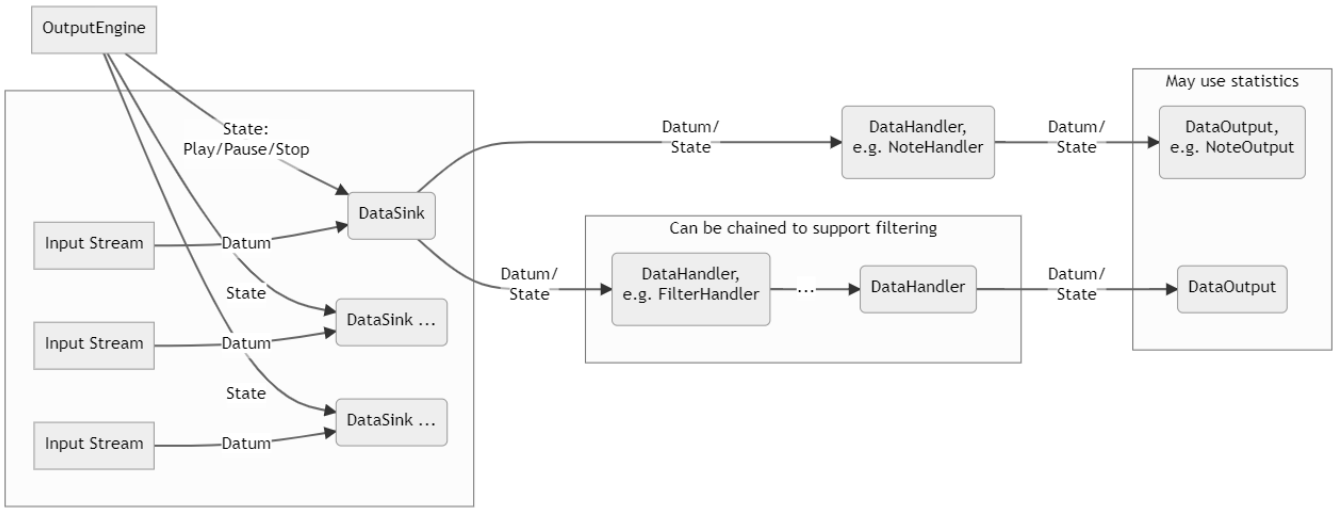


**Figure 1: Every object in PSST is an RxJs stream. By chaining objects of different types together, the user can specify the sonification of a stream. DataSinks merge together Datum objects with system state information, both of which are then shared with all downstream objects.**

`Observable` [64]. All internal communication also uses RxJs streaming. The primary role of a `DataSink` is to share both incoming data and state information (*e.g.,* stopped, paused) with all `DataHandlers`. They hold a single stream of timestamped numbers (specifically, streams containing instances of the `Datum` class are provided as an RxJs `Observable`).

If the incoming data is not already encapsulated in a Datum object, it is wrapped up with a timestamp at the time it arrives at the `DataSink`. We note that this allows us to easily support static data as well, by converting it to a stream of `Datum` using off-the-shelf RxJs features [64].

Data sinks stream to one or more `DataHandler` classes. A `DataHandler` should: (1) handle data based on its state (i.e., paused, playing, or stopped), (2) possibly manipulate the data, changing what any subscribers see, or filter data, preventing it from reaching any subscribers, and (3) output the data to its `DataOutput` displays, if relevant. `DataHandlers` can be chained, so a new `DataHandler` can either receive input from its `DataSink` or from an existing `DataHandler`.

Each data handler has one or more `DataOutputs` associated with it. These outputs can play specific audio files, speak values, play white noise, play a note corresponding to the value that needs to be outputted, and add elements to an SVG file.

For example, a `NotificationHandler` takes a range and an output appropriate for notification (such as speaking the value of a data point). When a `Datum` in the input stream is inside the range, it is passed on to the output, otherwise it is filtered out. Similarly, a `NoteHandler` takes a range of values that might occur in the input stream, and converts a `Datum`'s position in that range into the audible range. By default, it passes these converted values on to the `NoteSonify` output.

We illustrate an example of a user configuring PSST to hear data from a light sensor (sample code for this scenario is in Section A.1). The user first creates a `NoteOutput`, configured to use both speakers for output. The user then creates a `NoteHandler`, passing the expected range of sensor values ($[0, 1]$ in our example) and the `NoteOutput`. The `NoteHandler` will use this information to modify any incoming `DataObject` into an audible range before streaming it to the `NoteOutput`. Finally, the user creates a `DataSink` (which can receive input from a RxJS stream or by direct calls to `sink.next()`). This will receive the light sensor's values and stream them to the `NoteHandler`.

The relevant sequence of events for a single new `Datum` is illustrated in Figure 2. We see the `OutputEngine` streams "Play" followed by a new `Datum` to the `Datasink`. This in turn streams both to the `NoteHandler`, which modifies the value of the `Datum` into an audible frequency relative to its position in the range $[0, 1]$ and then streams both to the `NoteOutput`, which in turn plays a note for the user.

## 3.1 Library Components

The PSST Library is extensible and handles some standard needs identified through our literature review and empirically through development of the demos described in Section 4.

PSST is implemented in TypeScript. Audio output is handled using the Web Audio framework and speech is generated using the Web Speech API—both technologies built into modern web browsers. RxJS is used for all streaming-related functionality.

***Data Handlers.*** PSST includes *Data Handlers* (listed in Table 2) for filtering, tracking data patterns, and scaling. By default, Data Handlers support subscription by one or more Output objects.

The handlers provided fall into two main classes. *Transformations* modify the data without filtering it. For example, `ScaleHandler` uses a provided function to translate from an input *range* to an output *domain*. This is useful, for example, in its subclass, `NoteHandler`, which includes some specific parameters for the *domain* relevant to

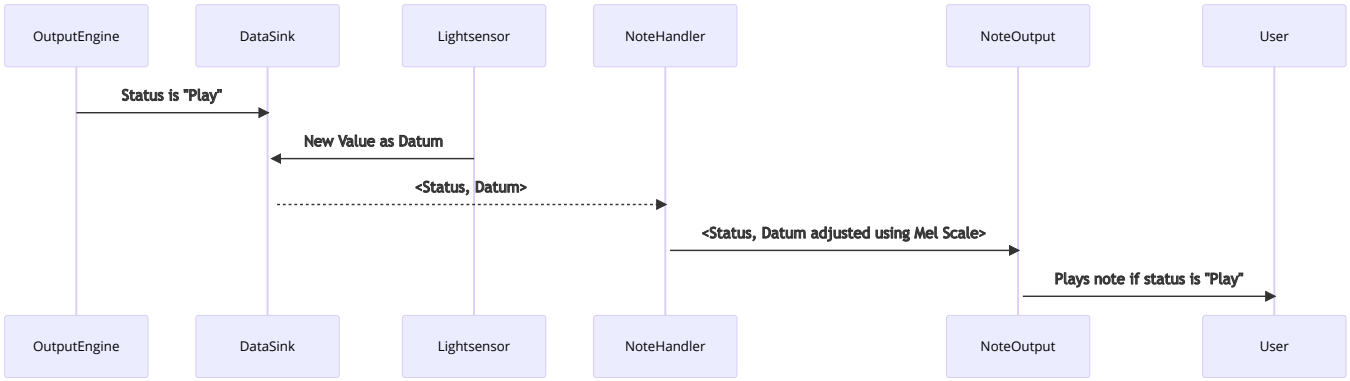


**Figure 2: A sequence diagram showing two events streamed to a DataSink – first, a Status, and second a new Datum. These are passed on to the NoteHandler, which adjusts the Datum value before passing it to the NoteOutput. The NotOutput plays the note if Status is "Play."**

| Classification | Handler Example | Description | What is Created |
|---|---|---|---|
| Transformation: Converts values from an input range to an output domain | `ScaleHandler` | Generic transformation function | Scaled values |
| | `NoteHandler` | Specifically maps properly to audible notes | Audible frequency values |
| | `SlopeParityHandler` | Indicates whether the line's slope has changed | Boolean values |
| Filter: Removes a subset of values based on query criteria | `FilterRangeHandler` | Removes values not in a specified range | Remaining values |
| | `ThrottleHandler` | Keeps every $n^{th}$ data point | Remaining values |
| | `NotificationHandler` | Keeps values that match a provided array | Remaining values |
| | `ExtremaHandler` | Keeps running extrema (maximum and/or minimum) based on all previous values | Most recent extreme value(s) |

**Table 2: Different default handlers supported by PSST. Users can extend this set by implementing new handlers.**

correct sonification as audible notes and a function that specifically handles the complications of audio. Finally, `SlopeParityHandler` returns true or false only when the slope has changed parity.

*Filters* modify which data points are seen downstream. Four handlers filter information: `FilterRange` handler only outputs `Datum` in a range; while `Notification` only outputs `Datum` that match an array of specific points. `Extrema` dynamically changes what is filtered based on what it has seen before (the previous maximum or minimum). Finally, `ThrottleHandler` reduces the volume of data by randomly dropping data points.

By chaining handlers together, additional possibilities arise. For example, `Extrema` followed by `SlopeParity` could be used to highlight whenever the data moves from a peak to a trough and vice versa.

***Outputs.*** Outputs are the leaf nodes in the data handling chain. They simply take data and display it. They do not know, or care, what handler they are attached to. PSST supports output to a frequency (using an oscillator), white noise, playing an audio file, and speech (Table 3). By default, the frequency output is continuous (*i.e.,* it plays until the next data point arrives) and all other outputs are of fixed duration. Frequency can be played in either the left or right channels or a combination, but does not currently support live panning. A new data point will interrupt the prior output if it arrives before the output is done; the user may configure the speech output to not interrupt.

| Output Example | Description |
|---|---|
| Note | Outputs a tone corresponding to the data. Can configure to play from the user's left, right, or both speakers. |
| Noise | Outputs white noise. Can use to highlight data after filtering. |
| Speech | Speaks the data value. Can configure the volume and rate of the speech. Useful after filtering and throttling. |
| Audio File | Outputs a wav file. Can choose a user-defined file name. Useful to alert to specific changes in the data. |
| SVG | Generates a graphical output. Can be used to generate output suitable for a hand-cranked music box. |

**Table 3: Outputs supported by PSST.**

***Statistics.*** Although not a core part of the architecture, another important abstraction supported by PSST is the concept of a "statistic." This is simply a function that can be applied to a `DataHandler`'s stream to calculate a summary statistic for that stream (*e.g.,* a running average), or provide a static summary statistic (*e.g.,* the expected max or min).

## 3.2 Dashboard Interface and Scenario

Although PSST is designed for programmatic use, a graphical user interface (GUI) that exposes a subset of the toolkit can reach additional users without requiring them to have programming experience. By encapsulating the PSST library as an accessible web application, the PSST dashboard interface (shown in Figure 3) enables BVI users to author sonification displays based on live sensor streaming data. We designed the web page with the aim to optimize accessibility via a screen reader and to support the design goals derived from our earlier literature survey (**DGs1-5**).

To support live streaming data, the interface provides connectivity with Jacdac microcontroller and sensor devices. Jacdac is a hardware and software stack that supports plug-and-play connections for low-cost hardware components and facilitates lightweight web development. With the PSST Dashboard, users can connect a Jacdac microcontroller, display live streaming data from attached sensors (*e.g.,* light sensors, accelerometers), configure sonification displays, and get immediate feedback by listening to sonified streaming data. Below, we explain how a BVI user can interact with the PSST Dashboard with an example scenario.

A user starts by connecting a Jacdac microcontroller via USB to their computer, and then selects the Connect button. Once the user selects the microcontroller driver from a modal dialog, the interface populates with all sensors connected to the microcontroller. In this case, the user has connected sensor devices that measure: light level, temperature, and humidity (Figure 3). The user interface also announces the connected sensors via an aria-live region. This ensures that whenever the user adds or removes a sensor to the microcontroller, their screen reader re-announces the changes. In the interface, each sensor stream is displayed visually with the most recently recorded value (*e.g.,* 72% for the light level, 24.2°C for temperature) and a button to add a `DataHandler` to that sensor data stream. This helps to support collaboration with sighted users.

The user can start the process for creating a sonification display by attaching a `DataHandler` to a sensor's stream. The user clicked the button to add a `DataHandler` to the humidity sensor after adding a `NoteHandler` to the light level sensor and a `FilterRangeHandler` to the temperature sensor (Figure 3).

To validate the extrema for a light sensor and calibrate it to the current ambient light level, the user can customize the sonification via the parameters and outputs for the attached `DataHandler`. In the case of the `NoteHandler`, which requires min and max values for a note range, the user provides 0 and 100, respectively. However, the user is not certain whether this is correct for the light level sensor (it will just play a very high or very low sound if they guessed the range wrong). To confirm the range, they add an `ExtremaHandler` and configure it to verbally speak the value of each `Datum` it encounters that is smaller than or equal to the current known minimum, or larger than or equal to the current known maximum (**DG2**).

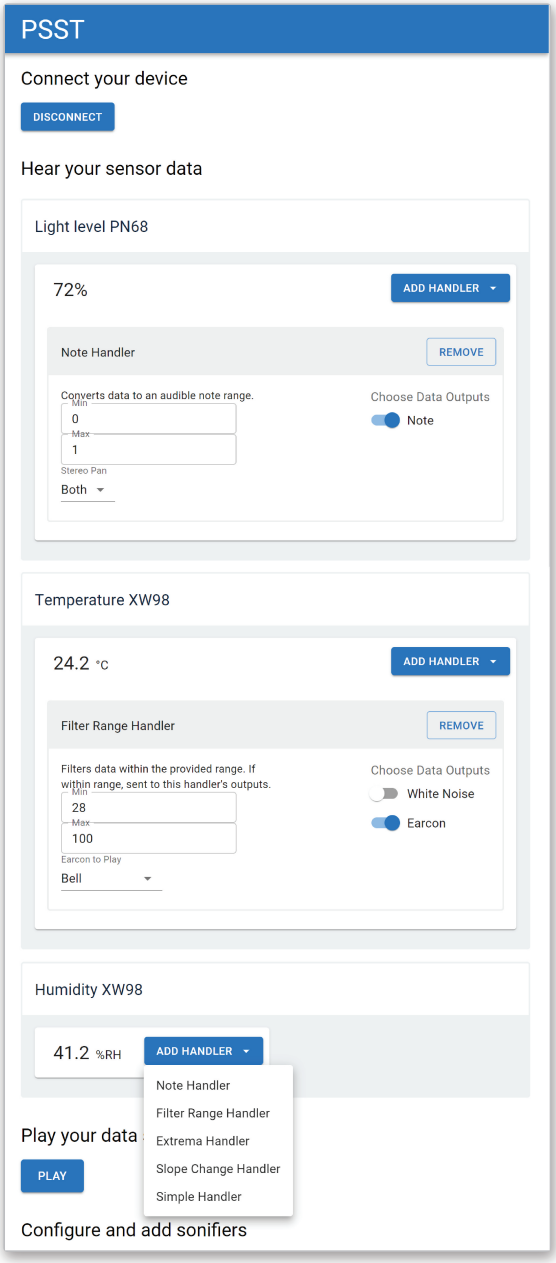


**Figure 3: The PSST dashboard interface for configuring sonification displays based on real-time sensor data.**

Once the user presses play (bottom of Figure 3), the system will sonify any incoming sensor readings based on the configuration. Now the user can play with the sensor and explore how it responds to physical stimuli (**DG1**). The user physically covers and uncovers the light level sensor with different objects, discovering which ones allow some light through. They confirm that total darkness produces a value of 0. They shine a bright flashlight light directly at the sensor and discover that the correct maximum is 1,

not 100 as they had previously thought (**DG3**). They can update the `NoteHandler` with this value dynamically in the interface and it will immediately adjust how it plays notes. The PSST dashboard supports real-time feedback of different sonification configurations (*e.g.,* adding and removing `DataHandlers`, or changing the parameters of a `DataHandler`).

Next, the user wants to investigate the temperature and humidity readings, both of which come from the same Jacdac sensor device. By holding the sensor over a steaming cup of tea, the user hopes to learn how closely the two readings are related. The user configures the `FilterRangeHandler` to have a range of 28 to 100°C and to have an earcon `DataOutput` that plays a "Bell" tone whenever the sensor data falls within that range. The user then adds a `SlopeChangeHandler` to the humidity data stream and configures its `DataOutput` to be an earcon that plays a "Whistle Down" tone. The user now wants the tone to only play when the humidity reading decreases over time (**DG4**), so they set the direction to be "Negative Only." Now, holding the sensor over the steaming cup of tea, the user hears a "Bell" tone to signify the temperature has reached 28°C. When the humidity decreases, they will hear a "Whistle Down" tone. A few moments after placing the sensor back on the table, the "Bell" tone has stopped and the "Whistle Down" tone finally emits from the speakers. From this, the user learns that some delay exists between the temperature and humidity readings (**DG1**).

# 4 VALIDATION

Our validation focuses on demonstrating the *utility* and *accessibility* of PSST. With respect to *utility* we demonstrate (1) the range of application spaces that a tool like PSST can support and (2) the relevance of PSST to the design goals shown in Table 1. With respect to *accessibility*, all of the demos were implemented by or with the collaboration of the blind first author of this paper; and we include feedback from two BVI developers who used the PSST dashboard, along with design choices, driven by first-person experience of the first author, that support accessibility in PSST. All of these goals are connected – for example, because of PSST's focus on supporting accessible displays of data, we include support for types of output that are not necessarily relevant to most sonification toolkits, which in turn increases the range of application spaces that PSST supports.

## 4.1 Range of Applications Spaces

We first demonstrate that PSST can handle a wide range of inputs and outputs. To this end, we implemented a physical data log that creates tangible output (and is compatible with a hand cranked music box); a piano interface that can be controlled with the keyboard; and replicated a feature of some screen readers, sonification of mouse position.

*A Physical Data Log.* To explore PSST's ability to support a wider variety of modalities, we created a `TangibleOutput` for PSST. This output generates an SVG with a history of data that it has seen over a fixed time period. The SVG can then be laser cut to create a tangible record of the data that can be explored tactily. Further, there is a genre of music box that takes as input a punchcard tape which specifies which notes to play, which this output is compatible with. This was done by extending the PSST library with a new output object containing approximately 25 lines of new code beyond the basic constructor.

*Keyboard and Mouse Input.* We implemented demos showing that keyboard and mouse input can be handled just like any other source of streaming data. For example, we convert keyboard letters to notes using a lookup table and they are straightforward to handle with a `NoteHandler`. Similarly we can map mouse position to notes using the stereo output capabilities of PSST combined with pitch. All of these are possible with no changes to the PSST architecture or library.

*Jacdac Input.* As described above, the PSST dashboard fully supports a wide range of sensors, including the accelerometer, light sensor, temperature and humidity sensor, sound level, and a potentiometer in the form of a physical slider.

We discuss this further in our exploration of the design goals, including a study demonstrating the value and accessibility of the PSST dashboard to two BVI participants.

## 4.2 Supporting Data Exploration Design Goals

As described in Table 1, we believe that a complete data display system for physical computing needs to support five key design goals: (1) learning how sensors respond to stimuli; (2) identifying appropriate thresholds for sensors; (3) being able to know about unexpected sensor readings; (4) analyzing trends and summarizing statistics; and (5) displaying multiple concurrent information streams. We provide examples that demonstrate PSST's value in supporting each of these goals next, and highlight which of these were used as tasks in our study.

*Learning how sensors respond to stimuli (DG1).* One of the basic needs for physical and embedded computing programming is making a sensor's state visible to understand how it responds to stimuli (DG1). An example of meeting this goal is in our scenario: using a light sensor to explore ambient light in a room. Another example is learning how the accelerometer values along each axis changes when it is rotated. We included both as tasks in our study.

*Identifying appropriate thresholds for sensors (DG2).* Whether exploring how a new sensor functions, calibrating it, or confirming its function in comparison to its description, a user may need to understand the minimum and maximum values the sensor is outputting. As an example, the first author of this paper learned something new about how an accelerometer worked when he first configured a sonification with a range of 0-1 and then discovered it was outputting a value of 2 when he shook it hard. In our study, participants are asked to explore the minimum and maxmimum values of a light sensor.

*Identifying unexpected readings (DG3).* A key goal to understand sensors is to identify expected and unexpected values that the sensor may output in specific scenarios. This can help with debugging and also discovery. For example, we asked participants to design a sonification that plays white noise when the sensor is placed flat on a table. A naive user might expect the accelerometer values to be unchanging when it is not being touched, but sonification should help the user to hear that this is not actually the case.

*Analyzing and summarizing data (DG4).* Transformations of data can be useful in deciding how and when to display it, or in providing new information that would otherwise be difficult to infer. This is especially important for BVI users as certain types of trends that are easily understood in the gestalt experience of viewing a chart might be less clear in the linear world of sound. As described earlier, it is straightforward to add new transformations to the PSST ecosystem, based on calculations that can take into account a single data point (as with `ScaleHandler`) or multiple data points (as with `SlopeHandler`). This was not used as a task in our study.

*Display multiple concurrent information streams (DG5).* When building physical computing systems, it is often the case that multiple sensors must work together in various ways. Understanding how two different sensors respond to the same stimuli, for example, can be important to building a more robust approach to recognizing when that stimuli is present. In the study, we asked participants to tell us how a temperature and a humidity sensor responded when placed at the mouth of a bottle filled with hot water.

## 4.3 Study with End Users

To explore the value of the PSST dashboard to end users, we recruited two BVI individuals and observed them using the dashboard. Our goals were to assess if the PSST dashboard successfully supports BVI people in authoring sonifications and answering questions about sensor data, and to understand how beginners could use PSST to learn about the sensors used in physical computing.

Toolkit evaluations with BVI developers in under-explored domains such as making and physical computing are likely to require careful consideration [45]. The study setup must be integrated with the right set of programming tools and a suitable definition of metrics such as task completion. In our evaluation using the PSST dashboard, we strove to avoid unnecessary accessibility barriers to ensure the best possible quality of data from our study.

*4.3.1 Study Setup: Method and Participants.* We started with a semi-structured interview. We asked participants to tell us about their experience with using sensors in programming, what worked, and things that they felt did not work well. We then introduced participants to the PSST dashboard and gave them the list of tasks mentioned above. We asked participants open-ended questions about each sensor involved in a task before and after the task. We then returned to interviewing, to find out what they liked. This uncovered confusions about the dashboard, and how PSST could be made more relevant to participants' own data-related needs. Each study was attended by 2 facilitators and a participant. We used a laptop running Windows 10, with both the JAWS and NVDA screen readers installed. Participants could use the screen reader of their choice, and could modify the screen reader configuration to suit their preference. All sessions were videotaped with two cameras and a software screen recorder to capture both the screen and the Jacdac hardware participants were interacting with. We reviewed researcher memos, notes taken during the study, and transcripts to identify themes, which we then discussed as a team.

We recruited two participants who self-reported as blind, with an advanced level of programming expertise, but with no physical computing experience. One participant used the NVDA screen reader and the other participant used JAWS.

*4.3.2 Results.* Initially, participants seemed skeptical about the relevance of physical computing / sensor-based programming. Their answers to questions about what they would like to learn about most of the sensors were somewhat rambling, unfocused and not very animated or excited. Even though participants had some understanding of sensors, it was not clear that the tasks seemed relevant to them. This contrasts strongly with their attitude after the study. As they explored the sensors they became increasingly engaged. P1 articulates this by saying:

> *"It was just kind of cool to see that for the first time in action I guess, with an actual tangible device... It was good to actually use them. I mean, you know, a lot of it, I think what might be true for a lot of blind people could be very theoretical things when you read about them but to actually kind of like see them in action when you don't really use all these things in an actual lab and to see how things change when you kind of experiment, I guess, that was cool."* (P1)

As further evidence of PSST's relevance beyond physical computing, both participants suggested new application domains for sonification. For example, one was interested in applying it to high volumes of telemetry data generated by applications. Another participant found value for this toolkit in data science work: *"You know, for example, kind of like in Pandas [a python data analysis library] if you can give a data frame or something and then, if I can actually set up these handlers through this library and then export or you know save the output into a file . . . and get a more refined data dump . . . that would be helpful."* (P1)

Below, we describe how they used the PSST dashboard, including their ideation process and its accessibility, and their engagement with the goals identified in Table 1.

*Use of the PSST dashboard.* For most tasks, participants used the note handler. The extrema handler in combination with the speech output was used to find the values of sensors related to specific stimuli. One of our participants misinterpreted the file output–an output that plays static audio files. They believed it would also vary the pitch of the selected audio file, similar to functionality offered by the note handler. As a result of this misunderstanding, the participant initially assumed that moving an accelerometer was not causing the sensor values to change, until we explained that file outputs do not change pitch.

Participants iterated on displays to gain new insights. They first ideated on what would work, and then tried customizing the display – there was no single way to answer each question for them. They seemed excited by the process. For example, P2 found the ability to customize what they hear to be very useful: *"The thing I like the most about this was the customizability of the handlers. That was really cool. I've seen these sonifications for decades . . . but I've never seen anything nice in a dashboard like this that lets you add and remove multiple ones and tweak the parameters of each."* (P2)

With respect to accessibility, both participants were able to use the dashboard but one requested that the dashboard use more HTML headings, and found the Speech Output too noisy. Before

the second participant session, we annotated each sensor's name with a HTML heading (to improve efficiency using a screen reader) and added the throttle handler to reduce the frequency of spoken output. One participant also wanted the ability to spatialize speech, an interesting opportunity to consider in the future.

In terms of the design goals included in the study (DG1-3 and DG5), participants demonstrated that we addressed them successfully. Below we provide some examples of participant feedback which illustrates that.

*Learning how sensors respond to stimuli (DG1).* P2 was interested to understand more about the sensitivity of different light sensors.

> *"Maybe you'd want to know how fast it responds to change, or something like that. Because I've seen that with sensitive light sensors, you're able to identify, say, the flickering of an electric bulb, compared with the constancy of sunlight, which I think is quite fascinating. Wagering whether you have a sensitive light sensor – I don't know – that could be something someone wanted to know. And honestly, I've never experienced that, I've just heard that."* (P2)

When we asked our participants to examine and tell us how a temperature and a humidity sensor responds when they place the sensor at the mouth of a bottle with hot water in it, our participants noticed that the humidity sensor's value was increasing at a slower rate than the temperature sensor. Both participants added two note handlers; one for each sensor, mapped to the left and right speaker. One participant commented that the humidity sensor's value was increasing in steps whereas the temperature sensor's value was increasing more linearly.

*Identifying appropriate thresholds for sensors (DG2).* When asked to locate the light sensor on a micro:bit and report the minimum value of the sensor when it is covered, our participants made observations that the light sensor was very small in size and was noisier than they expected it to be. After adding a note handler to hear how the sensor value was changing and an extrema handler to hear the new minima as they occurred when the user covered the light sensor, one participant also commented on the sensitivity of the sensor, noting that the sensor was much more sensitive than they expected it to be.

*Learn about unexpected sensor readings (DG3).* Our participants derived many insights about the accelerometer using PSST. To explore the accelerometer, we asked participants to tell us the axis that changes the most when the accelerometer is rotated in each of the three directions. One participant noted how, as the accelerometer was moved, the values from all three axes tended to change, although at any point in time, one of the axes was typically changing the most. We asked our participants to design a highlight that plays when the sensor is placed flat on the table. One participant specifically questioned his prior understanding of an accelerometer. *"the accelerometer doesn't behave as I expected... and this is to do with a blind guy's 3D geometry"* (P2). The same participant, for example, expected the accelerometer values to be 0 when the sensor is placed on a table. Additionally, he learned that the sensor was noisy when he heard different values being announced when the sensor was placed flat on the table.

*Display multiple concurrent streams (DG5).* Throughout the study, participants configured displays that streamed data from multiple sources. For tasks that required comparing data from multiple axes of the accelerometer, participants configured displays with note and speech outputs to sonify the different axes. For tasks that required participants to compare the temperature and humidity sensor, they configured multiple note handlers and outputs mapped to the left and right audio channels. Finally, our participants configured multiple handlers to the same sensor to understand *how* the sensor was responding, and *what* the value was. For example, our participant used both a note and extrema handler when trying to understand the light sensor; using information from the extrema handler he reported the change in value, and using information from the note handler he observed that the sensor was very sensitive because its value was changing even when he had his finger on it.

## 5 DISCUSSION

Our accessible PSST dashboard let developers use the toolkit and alleviated accessibility barriers that might have come up in a less structured study [56, 70]. Our evaluation of PSST with two BVI developers shows that PSST can help with understanding sensor data. The ability to easily and accessibly customize how data is presented differentiates our toolkit from other sonification libraries. We present limitations of PSST and our evaluation, and discuss the importance of performing toolkit evaluation with BVI developers.

In addition to demonstrating that PSST is accessible itself and also makes physical computing more accessible, our study shows that lack of access has a desultory effect on what people may think is interesting or worth doing. The excitement and engagement that developed over the course of our study echo the improvements that more relevant and accessible technologies have given other communities, such as the LilyPad Arduino's impact on the engagement of women in physical computing [11].

### 5.1 Limitations

Though PSST provides a tool and foundation for BVI developers to customize their access to data, we do not yet know the full extent to which PSST can be used in an end-to-end physical computing programming scenario. We only included two developers in our study, making any quantitative analysis of impact impossible. However, the change in attitude reported in the participants before and after the study gives a strong indication that the toolkit is meeting real needs among at least some subset of BVI developers.

In our validation, we did not study or evaluate use of the toolkit API with BVI developers. This means that we were not able to assess things like the difficulty of adding novel handlers, outputs, and statistics.

Finally, PSST's library is still relatively small and would benefit from additional outputs drawn from the rich sonification literature [32] to provide well-studied interpretable ways of presenting data in audio. For example, live panning and stereo output of all types of audio would be helpful additions to PSST. Furthermore, PSST does not currently support the ability to go back in time, or replay values after a pause.

### 5.2 Reflections on an Inclusive Design Process

The entire design process of PSST involved a Blind developer (the first author). This required us to scaffold the end-to-end development process. It was necessary to bootstrap the ability to understand data and sonification to develop new tools for understanding data and sonification. Our process involved building a simple user interface that was easily customized in code from the very beginning. Initially, this allowed the upload of CSV files with easily understood, pre-selected datasets in them for sonification using our earliest prototypes. On implementing new features of PSST, the author could debug PSST's sonification by testing with datasets that generated a sign wave, square wave, and sawtooth wave. Having a couple of known datasets, a simple UI that made available different types of static data, and the ability to configure sonifications through code, helped the lead author to explore sonifications with different types of data and configurations.

Our next step was to add limited support for a micro:bit accelerometer, one of the more complex sensors on that platform. The axis of the accelerometer to sonify could be selected, and one or more handlers and outputs that were pre-configured in code could be attached. The first author had never programmed using data from an accelerometer before, nor used multi-stream sonifications in programming before. Our process let us verify the usefulness of our approach and demonstrated the value of sonification in learning about sensors: for example, the first author used a combination of a slope parity handler and speech output to access the values of different accelerometer axes, and he, like P2, realized how noisy the sensor was even when lying still on a table. After implementing new features such as handlers and outputs for PSST, the author could use PSST to verify what he knew about the accelerometer. If the sonification did not match his expectation, the author would discuss the findings with the research team. These conversations helped to determine whether there was a bug with PSST, or a misunderstanding about how accelerometers work.

Finally, because our team involved both sighted and BVI people, we considered how we might support collaboration by showing information visually about what was being streamed. While basic visual feedback was valuable for collaboration, we did not in the end need to implement more sophisticated visual displays, as all team members could hear the sonification.

### 5.3 PSST in Other Domains

We designed PSST around anticipated needs to understand streaming physical computing sensor data by allowing BVI developers to customize how they want to access this data. As noted by our participants, however, PSST could be applied to a broad set of programming contexts such as telemetry, mobile computing, machine learning, and data science.

Extending PSST in this way would require similar attention to integration that we demonstrated with the PSST dashboard and the micro:bit. Prior work suggests that assembling an accessible programming workflow requires careful balance of social factors such as being in sync with tools collaborators might use [9], technical factors such as accessibility of the tool itself [2], and nuanced informational needs to perform a particular task [56]. Understanding the specific needs of other domains will improve the potential for PSST to support BVI programmers. An important next step would be to study these domains and properly customize PSST to interoperate with existing programming tools and processes within them.

## 6 CONCLUSIONS

We presented PSST, a streaming data sonification toolkit designed to make data from sensors accessible to BVI developers. We demonstrated the utility and capabilities of our toolkit by generating audio and physical logs of sensor data, and showed its applicability by sonifying the position of the mouse pointer and the keyboard. Our evaluation with two BVI developers shows that PSST's customizability was most valued by our participants, and the variety of handlers and outputs built into PSST provide tools for BVI developers to understand data from sensors, potentially helping them engage with physical computing.

Prior sonification toolkits have not historically been evaluated with BVI developers. Consequently, many efforts do not provide the flexibility needed for BVI people to customize sonification displays themselves, to suit their particular needs.

Our work is a valuable first step in the exploration and evaluation of toolkits in the under-explored domain of accessible authoring of sonification schemes for streaming sensor data. We hope that PSST and our dashboard will open up new opportunities for researchers to explore BVI authoring of data displays, and garner interest from a wider set of BVI developers to engage with physical computing.

## ACKNOWLEDGMENTS

This work was funded by CREATE, NSF FMitF 1836813, and Google.

## REFERENCES

[1] Samuel Aaron, Alan F Blackwell, and Pamela Burnard. 2016. The development of Sonic Pi and its use in educational partnerships: Co-creating pedagogies for learning computer programming. *Journal of Music, Technology & Education* 9, 1 (2016), 75–94.

[2] Khaled Albusays, Stephanie Ludi, and Matt Huenerfauth. 2017. Interviews and observation of blind software developers at work to understand code navigation challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, Maryland, USA) *(ASSETS '17)*. Association for Computing Machinery, New York, NY, USA, 91–100. https://doi.org/10.1145/3132525.3132550

[3] Jack Atherton and Paulo Blikstein. 2017. Sonification blocks: A block-based programming environment for embodied data sonification. In *IDC 2017 - Proceedings of the 2017 ACM Conference on Interaction Design and Children*. https://doi.org/10.1145/3078072.3091992

[4] Jonny Austin, Howard Baker, Thomas Ball, James Devine, Joe Finney, Peli De Halleux, Steve Hodges, Michał Moskal, and Gareth Stockdale. 2020. The BBC micro:bit: From the UK to the world. *Commun. ACM* 63, 3 (2020), 62–69.

[5] Catherine M. Baker, Lauren R. Milne, and Richard E. Ladner. 2015. StructJumper: A tool to help blind programmers navigate and understand the structure of code. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3043–3052. https://doi.org/10.1145/2702123.2702589

[6] Kirsty Beilharz and Sam Ferguson. 2009. An interface and framework design for interactive aesthetic sonification. In *Proceedings of the 15th International Conference on Auditory Display (ICAD 2009)*.

[7] Cynthia L Bennett, Abigale Stangl, Alexa F Siu, and Joshua A Miele. 2019. Making nonvisually: Lessons from the field. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. 279–285.

[8] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the problems of end-user developers in a physical computing task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*, Jofish Kaye, Allison Druin, Cliff Lampe, Dan Morris, and Juan Pablo Hourcade (Eds.). ACM, 3485–3497. https://doi.org/10.1145/2858036.2858533

[9] Stacy M. Branham and Shaun K. Kane. 2015. The invisible work of accessibility: How blind employees manage accessibility in mixed-ability workplaces. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility* (Lisbon, Portugal) *(ASSETS '15)*. Association for Computing Machinery, New York, NY, USA, 163–171. https://doi.org/10.1145/2700648.2809864

[10] J. W. Bruce and N. T. Palmer. 2005. SIFT : Sonification integrable flexible toolkit. In *ICAD 05-Eleventh Meeting of the International Conference on Auditory Display*.

[11] Leah Buechley and Benjamin Mako Hill. 2010. LilyPad in the wild: how hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM conference on designing interactive systems*. 199–207.

[12] Erin Buehler, Stacy Branham, Abdullah Ali, Jeremy J Chang, Megan Kelly Hofmann, Amy Hurst, and Shaun K Kane. 2015. Sharing is caring: Assistive technology designs on thingiverse. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 525–534.

[13] Xiang 'Anthony' Chen, Jeeeun Kim, Jennifer Mankoff, Tovi Grossman, Stelian Coros, and Scott E Hudson. 2016. Reprise: A design tool for specifying, generating, and customizing 3D printable adaptations on everyday objects. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 29–39.

[14] Juliana Cherston and Joseph A Paradiso. 2017. Rotator: Flexible distribution of data across sensory channels. In *International Conference on Auditory Display (ICAD)*.

[15] Chih-Chao Chung and Shi-Jer Lou. 2021. Physical computing strategy to support students' coding literacy: an educational experiment with arduino boards. *Applied Sciences* 11, 4 (2021), 1830.

[16] Josh Urban Davis, Te-Yen Wu, Bo Shi, Hanyi Lu, Athina Panotopoulou, Emily Whiting, and Xing-Dong Yang. 2020. TangibleCircuits: An interactive 3D printed circuit education tool for people with visual impairments. In *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*, Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik (Eds.). ACM, 1–13. https://doi.org/10.1145/3313831.3376513

[17] Alberto De Campo. 2007. Toward a data sonification design space map. Georgia Institute of Technology.

[18] Lieven De Couvreur and Richard Goossens. 2011. Design for (every) one: co-creation as a bridge between universal design and rehabilitation engineering. *CoDesign* 7, 2 (2011), 107–121.

[19] Daniel Drew, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. The Toastboard: Ubiquitous instrumentation and automated checking of breadboarded circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST 2016, Tokyo, Japan, October 16-19, 2016*, Jun Rekimoto, Takeo Igarashi, Jacob O. Wobbrock, and Daniel Avrahami (Eds.). ACM, 677–686. https://doi.org/10.1145/2984511.2984566

[20] Gaël Dubus and Roberto Bresin. 2011. Sonification of physical quantities throughout history: a meta-study of previous mapping strategies. International Community for Auditory Display.

[21] Kajetan Enge, Alexander Rind, Michael Iber, Robert Höldrich, and Wolfgang Aigner. 2021. It's about time: Adopting theoretical constructs from visualization for Sonification. In *AM '21: Audio Mostly 2021, Virtual Event / Trento, Italy, September 1-3, 2021*, Luca Turchet (Ed.). ACM, 64–71. https://doi.org/10.1145/3478384.3478415

[22] Benjamin Jotham Fry. 2004. *Computational information design*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[23] Ahmad Adamu Galadima. 2014. Arduino as a learning tool. In *2014 11th International Conference on Electronics, Computer and Computation (ICECCO)*. IEEE, 1–4.

[24] Beatriz Garcia, Wanda Diaz-Merced, Johanna Casado, and Angel Cancio. 2019. Evolving from xSonify: a new digital platform for sonorization. In *EPJ Web of Conferences*, Vol. 200. EDP Sciences, 01013.

[25] Alexandra Gendreau Chakarov, Quentin Biddy, Colin Hennessy Elliott, and Mimi Recker. 2021. The Data Sensor Hub (DaSH): A physical computing system to support middle school inquiry science instruction. *Sensors* 21, 18 (2021), 6243. https://doi.org/10.3390/s21186243

[26] William Gerrey. [n.d.]. The Smith-Kettlewell Technical File. A Quarterly Publication of The Smith-Kettlewell Eye Research Institute's Rehabilitation Engineering Research Center. https://www.ski.org/smith-kettlewell-technical-file.

[27] Gareth Halfacree. 2017. *The official BBC micro:bit User Guide*. John Wiley & Sons.

[28] Thomas Hermann, Andy Hunt, and John G Neuhoff. 2011. *The sonification handbook*. Logos Verlag Berlin.

[29] Tobias Hildebrandt and Stefanie Rinderle-Ma. 2013. Toward a sonification concept for business process monitoring. In *19th International Conference on Auditory Display ([ICAD] 2013)*.

[30] Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil, and Steven Johnston. 2013. .NET Gadgeteer: A new platform for K-12 computer science education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 391–396. https://doi.org/10.1145/2445196.2445315

[31] S. Hodges, S. Sentance, J. Finney, and T. Ball. 2020. Physical computing: A key element of modern computer science education. *Computer* 53, 04 (April 2020), 20–30. https://doi.org/10.1109/MC.2019.2935058

[32] Leona Holloway, Cagatay Goncu, Alon Ilsar, Matthew Butler, and Kim Marriott. 2022. Infosonics: Accessible infographics for people who are blind using sonification and voice. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3491102.3517465

[33] Amy Hurst and Shaun Kane. 2013. Making "making" accessible. In *Proceedings of the 12th international conference on interaction design and children*. 635–638.

[34] Amy Hurst and Jasmine Tobias. 2011. Empowering individuals with do-it-yourself assistive technology. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. 11–18.

[35] Khaled Hussein, Eli Tilevich, Ivica Ico Bukvic, and Soo Been Kim. 2009. Sonification design guidelines to enhance program comprehension. In *IEEE International Conference on Program Comprehension*. https://doi.org/10.1109/ICPC.2009.5090035

[36] Kritisha Kantilal Jain. 2021. *Making Makerspaces more accessible for people with visual impairment: Understanding user needs to reimagine solutions*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[37] Petr Janata and Edward Childs. 2004. Marketbuzz: Sonification of real-time financial data. In *International Conference on Auditory Display (ICAD)*.

[38] NW Kim, SC Joyner, A Riegelhuth, and Y Kim. 2021. Accessible visualization: Design space, opportunities, and challenges. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 173–188.

[39] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2004), 26-29 September 2004, Rome, Italy*. IEEE Computer Society, 199–206. https://doi.org/10.1109/VLHCC.2004.47

[40] Robert Kowalski. 2009. *Prototyping*. Number ISSN 1862-5207. Hauptseminar Medieninformatik WS, Chapter Prototyping in physical computing-Sketching in hardware, 87. Technical Report LMU-MI-2010-1.

[41] Qiao Lin, Yue Yin, Xiaodan Tang, Roxana Hadad, and Xiaoming Zhai. 2020. Assessing learning in technology-rich maker activities: A systematic review of empirical research. *Computers & Education* 157 (2020), 103944.

[42] Suresh K Lodha, John Beahan, Travis Heppe, Abigail Joseph, and Brett Zane-Ulman. 1997. Muse: A musical data sonification toolkit. In *International Conference on Auditory Display (ICAD)*.

[43] Suresh K Lodha, Catherine M Wilson, and Robert E Sheehan. 1996. LISTEN: sounding uncertainty visualization. In *Proceedings of Seventh Annual IEEE Visualization'96*. IEEE, 189–195.

[44] Stephanie Ludi. 2015. Position paper: Towards making block-based programming accessible for blind users. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*. IEEE, 67–69.

[45] Kelly Mack, Emma McDonnell, Venkatesh Potluri, Maggie Xu, Jailyn Zabala, Jeffrey Bigham, Jennifer Mankoff, and Cynthia L. Bennett. 2022. Anticipate and adjust: Cultivating access in human-centered methods. In *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022*, Simone D. J. Barbosa, Cliff Lampe, Caroline Appert, David A. Shamma, Steven Mark Drucker, Julie R. Williamson, and Koji Yatani (Eds.). ACM, 603:1–603:18. https://doi.org/10.1145/3491102.3501882

[46] Tara M Madhyastha. 1992. *A portable system for data sonification*. Master's thesis. University of Illinois at Urbana-Champaign.

[47] William McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and checking behavior of embedded systems across hardware and software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017, Quebec City, QC, Canada, October 22 - 25, 2017*, Krzysztof Gajos, Jennifer Mankoff, and Chris Harrison (Eds.). ACM, 299–310. https://doi.org/10.1145/3126594.3126658

[48] Janis Lena Meissner, John Vines, Janice McLaughlin, Thomas Nappey, Jekaterina Maksimova, and Peter Wright. 2017. Do-it-yourself empowerment as experienced by novice makers with disabilities. In *Proceedings of the 2017 conference on designing interactive systems*. 1053–1065.

[49] Wanda Liz Diaz Merced. 2013. *Sound for the exploration of space physics data*. Ph.D. Dissertation. University of Glasgow.

[50] Microsoft. 2022. Jacdac. https://aka.ms/jacdac

[51] Lauren R Milne, Catherine M Baker, and Richard E Ladner. 2017. Blocks4all demonstration: a blocks-based programming environment for blind children. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. 313–314.

[52] Cecily Morrison, Nicolas Villar, Alex Hadwen-Bennett, Tim Regan, Daniel Cletheroe, Anja Thieme, and Sue Sentance. 2021. Physical programming for blind and low vision children at scale. *Human–Computer Interaction* 36, 5-6 (2021), 535–569.

[53] M Nees and B Walker. [n.d.]. Auditory interfaces and sonication. *The Universal Access Handbook* ([n. d.]), 507–522.

[54] Michael A Nees. 2019. Eight components of a design theory of sonification. In *International Conference on Auditory Display (ICAD)*.

[55] Yoichi Ochiai. 2014. Visible breadboard: System for dynamic, programmable, and tangible circuit prototyping with visible electricity. In *International Conference on Virtual, Augmented and Mixed Reality*. Springer, 73–84.
[56] Maulishree Pandey, Vaishnav Kameswaran, Hrishikesh V. Rao, Sile O'Modhrain, and Steve Oney. 2021. Understanding accessibility and collaboration in programming for people with visual impairments. In *Proceedings of the CSCW Conference on Computer Supported Cooperative Work* (Virtual) *(CSCW '21)*. Association for Computing Machinery, New York, NY, USA, 30 pages.
[57] Sean Phillips and Andres Cabrera. 2019. Sonification workstation. In *International Conference on Auditory Display (ICAD)*.
[58] Venkatesh Potluri, Jennifer Manloff, James Devine, and Steve Hodges. 2021. Multisensory physical computing for the blind and visually impaired. In *CHI Workshop - Rethinking the Senses: A Workshop on Multisensory Embodied Experiences and Disability Interactions*.
[59] Venkatesh Potluri, Priyan Vaithilingam, Suresh Iyengar, Y. Vidya, Manohar Swaminathan, and Gopal Srinivasa. 2018. CodeTalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3174192
[60] Lauren Race, Chancey Fleet, Joshua A. Miele, Tom Igoe, and Amy Hurst. 2019. Designing Tactile Schematics: Improving Electronic Circuit Accessibility. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility, ASSETS 2019, Pittsburgh, PA, USA, October 28-30, 2019*, Jeffrey P. Bigham, Shiri Azenkot, and Shaun K. Kane (Eds.). ACM, 581–583. https://doi.org/10.1145/3308561.3354610
[61] Lauren Race, Claire Kearney-Volpe, Chancey Fleet, Joshua A. Miele, Tom Igoe, and Amy Hurst. 2020. Designing educational materials for a blind arduino workshop. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, CHI 2020, Honolulu, HI, USA, April 25-30, 2020*, Regina Bernhaupt, Florian 'Floyd' Mueller, David Verweij, Josh Andres, Joanna McGrenere, Andy Cockburn, Ignacio Avellino, Alix Goguey, Pernille Bjøn, Shengdong Zhao, Briane Paul Samson, and Rafal Kocielnik (Eds.). ACM, 1–7. https://doi.org/10.1145/3334480.3383055
[62] Lauren Race, Joshua A. Miele, Chancey Fleet, Tom Igoe, and Amy Hurst. 2020. Putting tools in hands: Designing curriculum for a nonvisual soldering workshop. In *ASSETS '20: The 22nd International ACM SIGACCESS Conference on Computers and Accessibility, Virtual Event, Greece, October 26-28, 2020*, Tiago João Guerreiro, Hugo Nicolau, and Karyn Moffatt (Eds.). ACM, 78:1–78:4. https://doi.org/10.1145/3373625.3418011
[63] A Roginska, E Childs, and M K Johnson. 2006. Monitoring real-time data: a sonification approach. In *Proceedings of the 12th International Conference on Auditory Display (ICAD2006)*.
[64] RxJS. 2022. Reactive Extensions Library for JavaScript. https://rxjs.dev/
[65] Emmanuel Schanzer, Sina Bahram, and Shriram Krishnamurthi. 2019. Accessible AST-based programming for visually-impaired programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 773–779. https://doi.org/10.1145/3287324.3287499
[66] JooYoung Seo and Gabriela T Richard. 2021. SCAFFOLDing all abilities into Makerspaces: A design framework for universal, accessible and intersectionally inclusive making and learning. *Information and Learning Sciences* (2021).
[67] Ather Sharif, Sanjana Shivani Chintalapati, Jacob O. Wobbrock, and Katharina Reinecke. 2021. Understanding screen-reader users' experiences with online data visualizations. In *ASSETS '21: The 23rd International ACM SIGACCESS Conference on Computers and Accessibility, Virtual Event, USA, October 18-22, 2021*, Jonathan Lazar, Jinjuan Heidi Feng, and Faustina Hwang (Eds.). ACM, 14:1–14:16. https://doi.org/10.1145/3441852.3471202
[68] Richard J Smythe. 2021. Realtime Data Plotting and Visualization. In *Advanced Arduino Techniques in Science*. Springer, 161–171.
[69] Katherine Steele, Brianna Blaser, and Maya Cakmak. 2018. Accessible making: Designing makerspaces for accessibility. *International Journal of Designs for Learning* 9, 1 (2018), 114–121.
[70] Kevin M. Storer, Harini Sampath, and M. Alice Merrick. 2021. "It's just everything outside of the IDE that's the problem": Information seeking by software developers with visual impairments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, 12 pages. https://doi.org/10.1145/3411764.3445090
[71] Evan Strasnick, Maneesh Agrawala, and Sean Follmer. 2017. Scanalog: Interactive design and debugging of analog circuits with programmable hardware. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017, Quebec City, QC, Canada, October 22 - 25, 2017*, Krzysztof Gajos, Jennifer Mankoff, and Chris Harrison (Eds.). ACM, 321–330. https://doi.org/10.1145/3126594.3126618
[72] Stephen Taylor. 2017. From program music to sonification: Representation and the evolution of music and language. Georgia Institute of Technology.
[73] Takahiko Tsuchiya, Jason Freeman, and Lee W Lerner. 2015. Data-To-Music API: Real-time data-agnostic sonification with musical structure models. *Proceedings of the 21st International Conference on Auditory Display (ICAD 2015)* (2015).
[74] Takahiko Tsuchiya, Jason Freeman, and Lee W. Lerner. 2016. Data-Driven live coding with Data-To-Music API. *Proceedings of the International Web Audio Conference*.
[75] Bruce N Walker and Michael A Nees. 2005. An agenda for research and development of multimodal graphs. In *International Conference on Auditory Display (ICAD)*.
[76] David Worrall, Michael Bylstra, Stephen Barrass, and Roger Dean. 2007. Sonipy: The design of an extendable software framework for sonification research and auditory display. In *Proceedings of the 13th International Conference on Auditory Display ICAD2007*.
[77] Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. 2017. CurrentViz: Sensing and visualizing electric current flows of breadboarded circuits. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017, Quebec City, QC, Canada, October 22 - 25, 2017*, Krzysztof Gajos, Jennifer Mankoff, and Chris Harrison (Eds.). ACM, 343–349. https://doi.org/10.1145/3126594.3126646
[78] Anat Zait. 2018. 4 simple steps for debugging your Arduino project. https://www.circuito.io/blog/arduino-debugging/

# A APPENDICES

## A.1 Sample Code Showing PSST Use

```
/**
 * Create a NoteOutput and configure
 * it to play audio from both speakers
 **/
output = new NoteOutput(0)

/**
 * Create a NoteHandler and specify the
 * range of input values will be between 0
 * and 1. By default, NoteHandler converts
 * values in this range to an audible range
 * using the Mel scale.
**/
handler = new NoteHandler([0, 1], output)

/**
 * Register a new sink with the OutputEngine
 * and add the NoteHandler to it
**/
lightSensorSink = OutputEngine.getInstance().
    addSink('lightSensorSink')
lightSensorSink.addDataHandler(handler)

/**
 * Send an OutputStateChange.Play downstream
 * to turn sonification on. Similar
 * calls can Pause and Stop play.
**/
OutputEngine.getInstance().next(
    OutputStateChange.Play)
/**
 * Any number of methods can be used to
 * pass on sensor data by calling
 * lightSensorSink.next([id],[sensorvalue])
 **/
```