ABSTRACT
During the development of physical computing devices, physical object models and programs for microcontrollers are usually created with separate tools with distinct files. As a result, it is difficult to track the changes in hardware and software without discrepancy. Moreover, the software cannot directly access hardware metrics. Designing hardware interface cannot benefit from the source code information either. This demonstration proposes a browser-based IDE named f3.js that enables development of both as a single JavaScript code base. The demonstration allows audiences to play with the f3.js IDE and showcases example applications such as laser-cut interfaces generated from the same code but with different parameters. Programmers can experience the full feature and designers can interact with preset projects with a mouse or touch to customize laser-cut interfaces. More information is available at http://f3js.org.

Author Keywords
Integrated development environment; personal fabrication; laser-cut interface; microcontroller.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces – GUI; D.2.6. Software Engineering: Programming Environments – Integrated environments.

INTRODUCTION
Physical computing, or the development process of the so-called Internet of Things (IoT), involves iterative cycles of modeling hardware and programming software. The recent proliferation of personal fabrication techniques reduces the cost and time of printing physical objects. Microcontrollers are becoming more powerful, and some are capable of running programs written in dynamically-typed languages such as JavaScript, allowing more software enthusiasts to create their prototypes easily.

Development of hardware and software is usually done with different tools, making it cumbersome to keep track of both changes synchronously. Moreover, tool separation prevents the software from accessing the hardware metrics and vice versa. For instance, modeling an object that has holes to support sensors and actuators requires manual input of their metrics. The source code contains 'import'-like statements for their drivers and thus might know which sensors and actuators are used, but it is not aware of the metrics, such as the size of the printed objects.
FORM FOLLOWS FUNCTION() – F3.JS
The main interface of f3.js is rendered as a single web page as shown in Figure 1. The top of the page presents general information (such as the project directory) and becomes more detailed as the page progresses. While common commands are provided as links under the Operations menu in the file manager, the user can always switch to Console or Secure Shell tabs and input any command.

The current system supports Tessel.io\(^1\) and Intel Edison\(^2\) microcontrollers, which can run Node.js-based JavaScript applications. Once the user chooses a directory on the system, existing code in the project is analyzed, and its overview is shown in the ‘Project Summary’ tab. While most of the information is specified in package.json and can be edited in-place, the platform type is automatically detected by analyzing the JavaScript source code.

The current f3.js library contains Node.js packages that drive Grove and Tessel.io modules. These are sensors and actuators encapsulated with standardized connectors, easily attached to the supported microcontrollers. The next tab ‘Code with Print Preview’ shows the source code editor and the print preview. The programmer can write JavaScript application code that imports the f3.js library and creates the hardware module instances on the code editor.

The f3.js library does not only enable controlling the hardware modules but also provides the metrics information and supports live programming of laser-cut interfaces. The print preview is dynamically rendered by executing the source code with the sandboxed f3.js JavaScript interpreter on the browser. The interpreter ignores any calls to undefined functions meant to run on the microcontrollers. When a change is made to the source code, the system instantly evaluates the code and updates the preview. With this live programming support, the programmer can easily design the laser-cut interface as well as the interactive GUI on the print preview. This interactivity, end-users including designers can use a mouse or touch to customize the laser-cut interfaces to fit their needs.

When the user is satisfied with the source code and the laser-cut interface, the system generates a PDF file and allows the user to laser-cut the acrylic panels. Finally, the assembled hardware can be connected to the host computer, and the user can upload the source code to start running the program on the microcontroller. The f3.js library on the microcontrollers does the same calculation as on the browser but does not render anything nor provide any interactivity (since there is no canvas to draw graphics) — therefore only allows to retrieve the metrics information. During execution of the program, f3.js shows a console with which the user can monitor the console output and input commands if needed.

RELATED WORK AND DISCUSSION
There exists prior work on programming tools for either physical objects or microcontroller applications. The former work includes DressCode \(^1\) that allows designers to write code in a domain-specific language that generates two-dimensional artifacts. ShapeJS \(^2\) provides JavaScript API that enables generating three-dimensional printer-ready objects. Our interaction design does not only allow the design of the hardware but also its behavior and can be integrated into these techniques. Recent work from the latter category includes .NET Gadgeteer \(^3\) and Autodesk 123D Circuits \(^4\), both of which support programming microcontrollers along with the graphical representations of the circuits. While these environments only care logical purposes, and has a defacto standard package manager, our system provides print preview that can be interactively edited by the designers and outputs laser-cut interfaces.

As discussed above, the novelty of our work resides in the integration of hardware and software design using a single programming language and single code base. The potential benefits include fast iterative development of applications aware of hardware metrics, simple version control, loose learning curve, and easy collaboration, whose validations remain as our future work. Whether JavaScript is suitable for designing layout of physical objects or not is an open question. Upon the choice of the language, we attached importance to the fact that it is popular, used for various purposes, and has a defacto standard package manager.

Further details on the f3.js IDE and library are available on http://f3js.org.

REFERENCES
2. Shapeways. ShapeJS. http://shapejs.shapeways.com