TypeScript: From JavaScript to Blockly and Back

Thomas Ball
Microsoft Research

Peli de Halleux
Microsoft Research

Sam El-Husseini
Microsoft

Richard Knoll
Microsoft

Michal Moskal
Microsoft Research

Abstract

While there are many JavaScript libraries for building solutions for a wide range of problems, it's not easy for novices to harness their power. We show how TypeScript, a gradually typed superset of JavaScript, can be used to bridge the gap between JavaScript and Blockly, a framework for creating block-based programming environments that greatly reduces the potential for syntax and semantic errors. In particular, we define a mapping from TypeScript to Blockly that makes it simple to create a domain-specific Blockly editor for a JavaScript library via a TypeScript declaration file. This mapping is supported by Microsoft MakeCode (https://makecode.com). An online editor for exploring the mapping from TypeScript to Blockly is at https://makecode.com/playground.

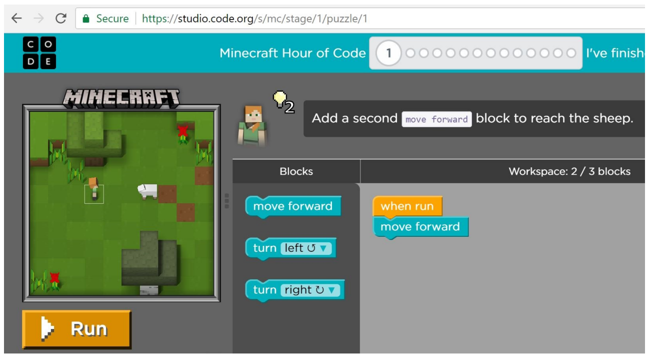
1 Introduction

Introducing programming to beginners used to be a process fraught with accidental complexity, due to the need to install toolchains and IDEs, typically created with the professional developer in mind. Today, the web has made available a plethora of programming environments; most any programming language can be experienced via the web. Some popular examples include:

- **JavaScript**: https://jsfiddle.net/;
- **Python**: https://www.learnpython.org;
- **C, C++, Ruby, ...**: https://repl.it/.

Not surprisingly, all the above approaches still use text editors for coding; this means that the possibility of introducing errors is still great, especially for beginners who don't understand the language. Features such as step-by-step tutorials and autocomplete can improve the experience, but the starting point still is not the most welcoming.

The "Hour of Code" experience,[1] has introduced over three hundred million students to coding,[2] with the express purpose of demystifying and breaking stereotypes about coding. In order to reach this many people, code.org used Google's Blockly,[3] a JavaScript framework inspired by MIT's Scratch



Add a second move forward block to reach the sheep.

Figure 1. Hour of Code example.

programming environment [1], to provide a very simple beginning programming experience, free of the possibility of syntax errors.

Figure 1 shows a screen snapshot from one of the many "Hour of Code" tutorials, in the family of "maze solving" problems. In this example, the goal is to program the Minecraft avatar "Steve" on the left of the tiny 2D world to move to the sheep on the right. To create a program, the user selects from a simple palette of three blocks (shown under the "Blocks" heading) and drags these blocks into the workspace. In this very simple example, the program is simply a sequence of commands. In later examples, concepts such as loops and conditionals are introduced.

In Blockly, colored blocks represent structured control-flow statements such as loops and if-then-else conditionals, as well as program expressions, values, and variables. Blocks also represent function calls to domain-specific APIs, via Blockly's support for *custom blocks*.[4] In the case of Figure 1, the domain-specific APIs are the commands for moving "Steve". Blockly can be compiled to a variety of languages, the main target being JavaScript, as Blockly itself is written in JavaScript and hosted in a web browser.

As the Blockly documentation states, "Blockly is a complicated library targeted at experienced developers", using both the XML format and JavaScript APIs for creating custom blocks programmatically.

[1]https://hourofcode.com/learn
[2]https://code.org/about/2016
[3]https://github.com/google/blockly

[4] https://developers.google.com/blockly/guides/create-custom-blocks/overview

BLOCKS+2018, Nov. 4, 2018, Boston, MA
2018.

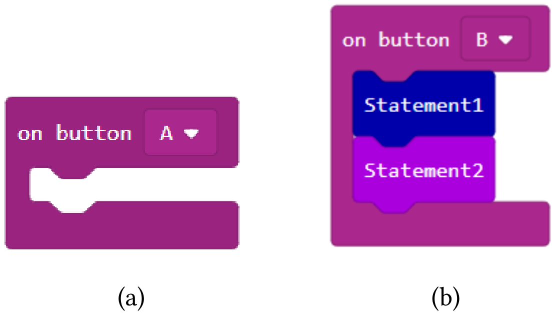


Figure 2. Block for event handling.

Our goal is to make it easier to bring the Blockly editing experience to more domains, leveraging the numerous JavaScript libraries available on the web, but without needing to become experienced with the Blockly framework.

TypeScript (www.typescriptlang.org) is a superset of the JavaScript language that is gradually typed, meaning that types may optionally be added to JavaScript code to provide for a more productive programming and debugging experience. Many JavaScript frameworks provide TypeScript declaration files, as can be found at the Definitely Typed GitHub repo.[5]

Our contribution is to describe a mapping from TypeScript annotated JavaScript APIs to Blockly that greatly simplifies the process of making JavaScript code available via Blockly.

It is not our goal to map every TypeScript programming construct into Blockly, as a one-to-one mapping of the TypeScript abstract syntax tree (AST) nodes to blocks would provide no abstraction, exposing the beginner to a visual representation of TypeScript in its full glory.

Rather, the goal is to support common programming paradigms with a simple mapping from TypeScript to blocks and back. This mapping exposes many TypeScript elements (eg., classes, nested functions, etc.) as uneditable blobs, while certain idioms (eg., **for** (let i = 0; i < expression; ++i)) are exposed as specialized blocks. This supports Blockly's goal to provide a simplified programming experience with higher-level abstractions.

2 Example

Below is an example of a TypeScript function *onButton*, with parameters b and f, that represents a common JavaScript pattern of registering an event handler (f) to be executed when some event occurs (in this case, a press of button A or button B, as specified in the enumeration *Button*):

```
export enum Button { A, B };
//% block="on button $b"
export function onButton(b: Button,
                         f: () => void): void { }
```

[5]https://github.com/DefinitelyTyped/DefinitelyTyped

The function *onButton* is explicitly typed using TypeScript's support for type annotations: each parameter has a type, which follows the colon; furthermore, the return type of the function also is specified as void.

The comment annotation specifies that a block B should be created for the function and that the text of the block should be "on button", as shown in Figure 2(a), with a single in-line parameter for the enumeration parameter b. The function definition and its types define the block's two *inputs*:

- a *value* input corresponding to the parameter b which allows selection of one of two possible parameter values, either "A" or "B", corresponding to the the two possible values for the enumeration *Button* — while an enumeration is realized as a JavaScript number, a Blockly field selector for this parameter only allows the user to choose either "A" or "B" (0 or 1);
- a *statement* input corresponding to the parameter f that allows the user to drag statement blocks inside of block B to define the body of the lambda function passed to f.

By default, a function that takes a function as its last parameter will give rise to a block with a statement input. Handling multiple function arguments is a direction for future work.

Figure 2(b) shows the event handler block with two blocks inside it, after an editing session by the user. The translation of the event handler block into TypeScript will yield the following code:

```
onButton(Button.B, function() {
    Statement1;
    Statement2;
})
```

3 Language Mapping

Users can explore the various mappings between TypeScript and blocks supported by MakeCode at https://makecode.com/playground, including namespaces, functions, classes, and factories. MakeCode also has a variety of field editors to make it easier to input data for particular types.

MakeCode supports compilation of Blockly to TypeScript and decompilation of TypeScript to Blockly.[6]

See MakeCode Labs (https://makecode.com/labs) for a set of example MakeCode editors created using the TypeScript/Blockly mapping.

References

[1] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay S. Silver, Brian Silverman, and Yasmin B. Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67. http://doi.acm.org/10.1145/1592761.1592779

[6] See https://github.com/Microsoft/pxt/tree/master/pxtblocks and https://github.com/Microsoft/pxt/tree/master/pxtcompiler/emitter.