User-defined Blocks for MIT App Inventor

Jane Im

Supervisor: Hal Abelson, MIT App Inventor
6.100 Project Report, Spring 2017

1. Introduction

MIT App Inventor is an open source web platform that lets users build their own Android Apps quickly by using blocks-based programming. MIT App Inventor currently has diverse types of components, from general blocks to specific task oriented components like sensors. However, it does not provide every type of block that an user needs. For example, currently there are no components for intricate graph visualization in App Inventor. However, there is a limitation to keep adding new categories of blocks, not only because the complexity of maintaining the system will increase, but also because an individual user's need is necessarily not a general need for others. In order to keep lowering the barriers of creating special goal oriented apps and to foster the learning of computational thinking, while also preventing the situation of adding every possible block to the system, a new feature called "user-defined blocks(customizable blocks)" has been implemented during 2017 Spring semester.

2. Definition of User-defined Blocks

User-defined blocks indicate the format of a block that could be customized to any kind of block the user wants to create, under the assumption that it 1.calls a function of a JavaScript API that is being used in the web page shown in the WebViewer component, and 2.has the right

format that corresponds to the function it triggers. It could also indicate any kind of block that a user has created using this format.

3.Background: WebViewer Component

WebViewer is a component in MIT App Inventor that lets users view web pages on their screens. There was previous work in MIT App Inventor to enable users to create and call JavaScript functions or variables with this component. The work on user-defined blocks is based on this previous work, to let users call any JavaScript functions that are currently being used in the web page that the WebViewer's url is referencing to.

4. Scenario

The scenario in this section will help the user's understanding of this paper. Jennifer is in charge of keeping track of her friends' scores of a game they regularly play. In order to track scores easily whenever she wants to, she decides to build a simple app that shows a table of her friends' names and scores using MIT App Inventor. She goes to App Inventor, takes out the WebViewer component, and follows the tutorial for making a simple app in which users could manipulate a table easily. Even though there aren't table related blocks in App Inventor, Jennifer creates five blocks that could each add a column, remove a column, add a row, remove a row, and set data of a table cell, using user-defined blocks. By using the blocks she created, she finally implements an app like *Figure 1*. In this app the user could type in a name or number to add or remove row or column, or set a data to a table cell. For example, in *Figure 1*, Jennifer typed in "Natalie" in the textbox for a column name, and pressed "ADD COLUMN" to add a column for her friend Natalie in the table.

Figure 1. App created using user-defined blocks, in which users can manipulate a table.

5. System Architecture

5.1 JSON Blocks

The first step was to add a new property called “JSONBlocks” to WebViewer. JSONBlocks is a long string of JSON format showing an array of objects. Each object conveys information of each block that the user wants to create and add to the WebViewer’s flyout. In each object, basic information of the block along with two newly added attributes are contained. The two newly added attributes are “function_name” and “parameters”, which are closely shown in *Figure 3*.

The “function_name” attribute indicates the name of the function that the block is calling from the JavaScript API that is used in the web page of the WebViewer. For example, if a block called “set_data” triggers a function called “setData”, the “function_name” of the object should be “setData”.

The section of parameters inform the system about which argument of the block corresponds to a particular parameter. If there are three parameters each corresponding to a row index, column index, and a value the table cell will be set to, the parameter section should each

sequentially contain information about the argument of the block for a row index, column index, and lastly the value a table cell will be set to, as it is shown in *Figure 3*. The "type" feature is especially important, since depending on the type of the argument, the system will run different code to extract the value from the block's argument to put in the JavaScript function as parameters. For example, if the corresponding argument is a field, the function "getFieldValue" will be called. Currently the system only differentiates inputs and fields, but since there are diverse types of inputs, and especially fields, the system will have to be modified to make more intricate differentiations.

When the user puts in a string of an array containing *n* objects that each convey information of an user-defined block and moves on to the "Blocks" section, one can see that *n* new blocks have been added to the WebViewer's flyout (*Figure 4*). The system has taken the long string that was in JSONBlocks property textbox and parsed it to extract information of each new block to add.

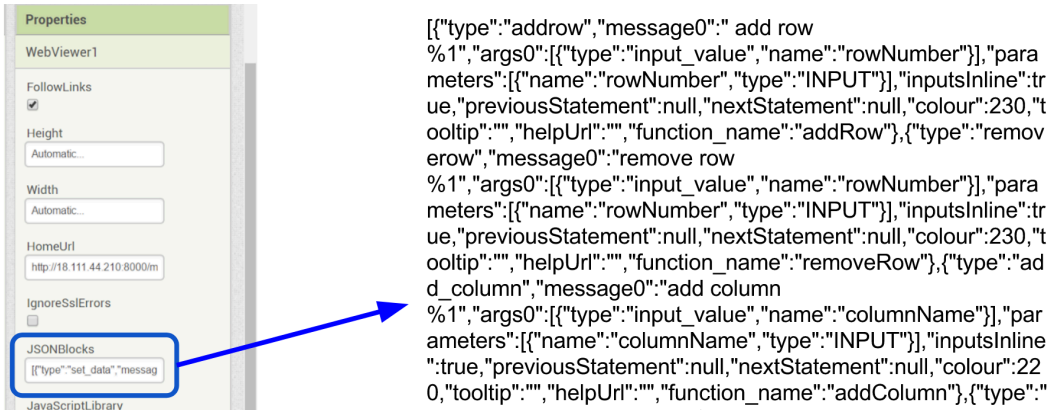

Figure 2. New property "JSONBlocks" added to WebViewer component

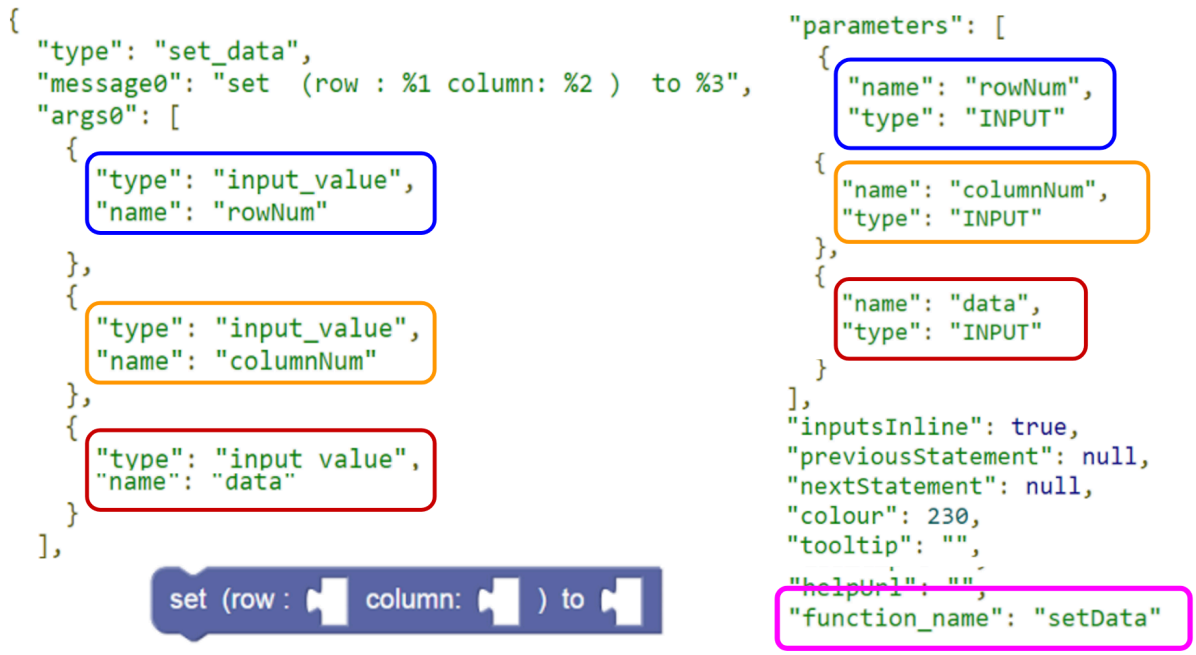

Figure 3. Example of an object for a user-defined block that calls function "setData"

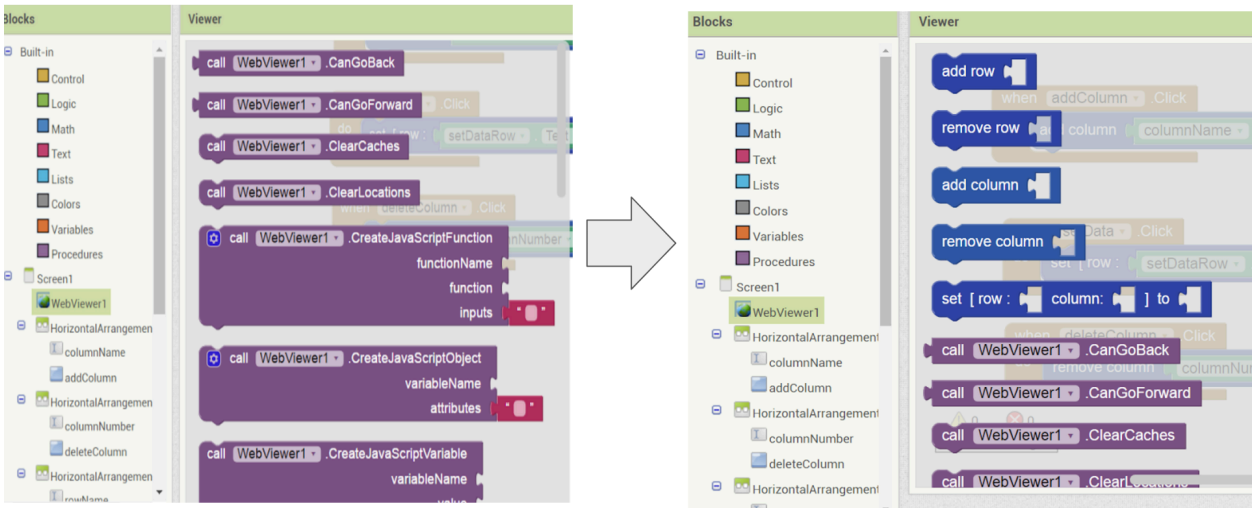

Figure 4. Effect of adding user-defined blocks

5.2 Blockly Developer Tools

Considering the aspect of usability, it is crucial to prevent the user from typing long sequences of objects to generate a string that needs to be pasted in JSONBlocks property.

Therefore, Blockly Developer Tools, a tool provided by Google to easily create blocks, was modified to create the right interface for this task.

Users can use Blockly Developer Tools to create a format of a block by easily snapping smaller pieces of blocks, such as blocks for input or field. Two new features “method name” and “parameters” were added to the root block as shown in in *Figure 5*. In “method name”, users type in the name of the function in the JavaScript API this block will trigger. In “parameters” users drop in a parameter block (*Figure 7*) and type in the name of the argument whose value will be put in as the corresponding parameter. Whether the type of an argument corresponding to each parameter is an “input” or a “field” should also be chosen , since different code is triggered according to each type.

Every time the user makes an update to the root block, the string that describes the block will be updated on the right, as shown in *Figure 6*. Another convenient aspect of Blockly Developer Tools is that every block that the user has created can be saved and stored in Block Library, as shown in *Figure 8*. After users have created all blocks, they can click “Get JSON String” button to see an alert box pop up with a selected string of the right format to put in the JSONBlocks property. The string includes all information of the blocks stored in Block Library, so the users can just copy the string and paste it to the JSONBlocks property textbox.

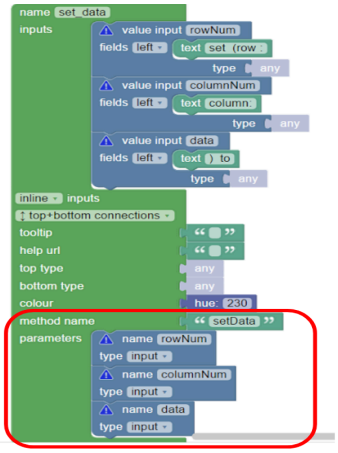

Figure 5. Two new attributes, "method name" and "parameters"

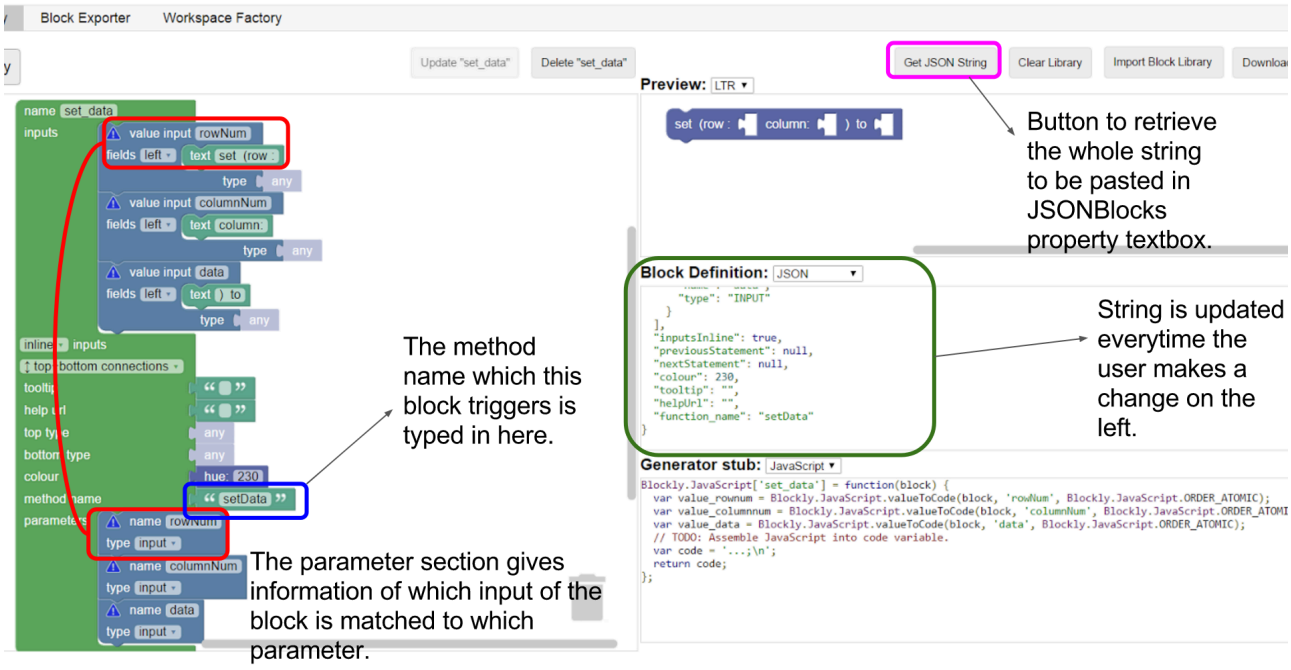

Figure 6. Modification of Blockly Developer Tools

Figure 7. Newly added parameter block. Type of "input" and "field" are distinguished.

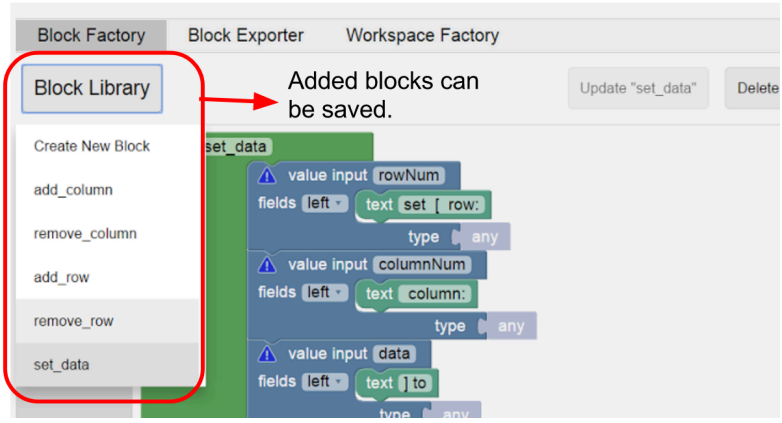

Figure 8. Newly added blocks can be saved.

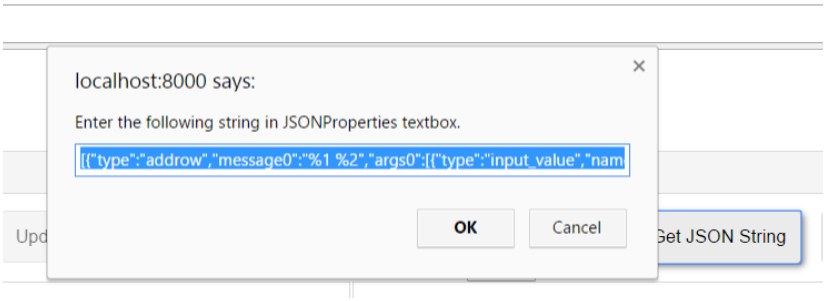

Figure 9. Users can easily copy and paste the string to JSONBlocks property

6.Demonstration

This section gives more details on the table manipulating app that was introduced in **4. Scenario**. The user would first have to drag and place the WebViewer component and set the HomeURL to the address of the web page showing the page of the basic table.

In order to create user-defined blocks that will manipulate the table, the user first has to make one JSON formatted information for each block, following the process described in **5.2** by using Blockly Developer Tools. In this case, the user makes one for five blocks, and clicks “Get

JSON String" to copy one big string containing all five information, and pastes it into the JSONBlocks property textbox.

In the designer section, the user also has to drag and drop text boxes where the user will be typing in values(when using the app) that will be put in the user-defined blocks, and later plugged into the corresponding JavaScript functions. Users will also have to add buttons that will each trigger a user-defined block which will then call the corresponding JavaScript function. For example, in *Figure 10*, the value in textbox "columnName" will be put in as an input of the "addColumn" function which the user-defined "addColumn" block triggers, after "Add column" button is pressed.

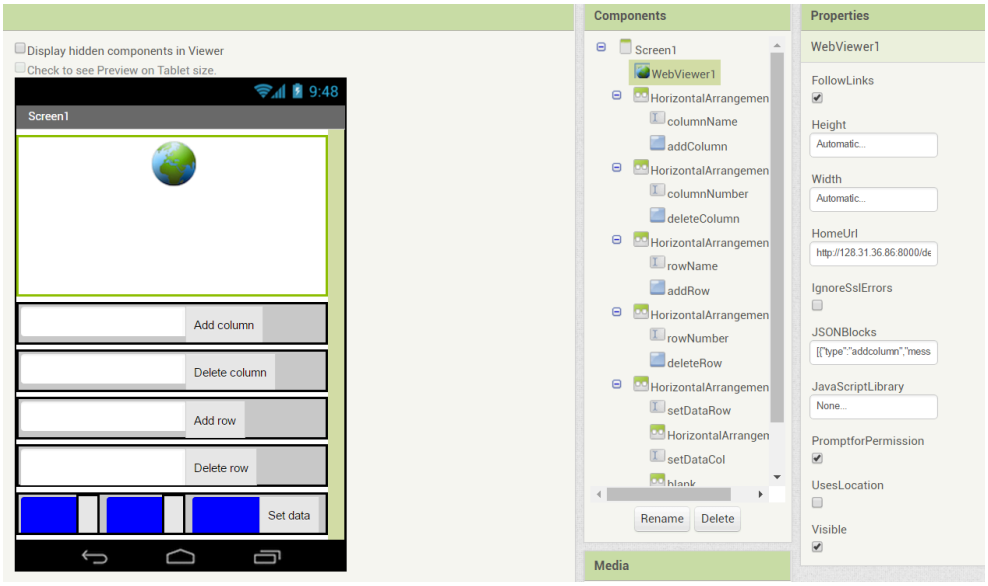

Figure 10. The designer section for making the table manipulating app

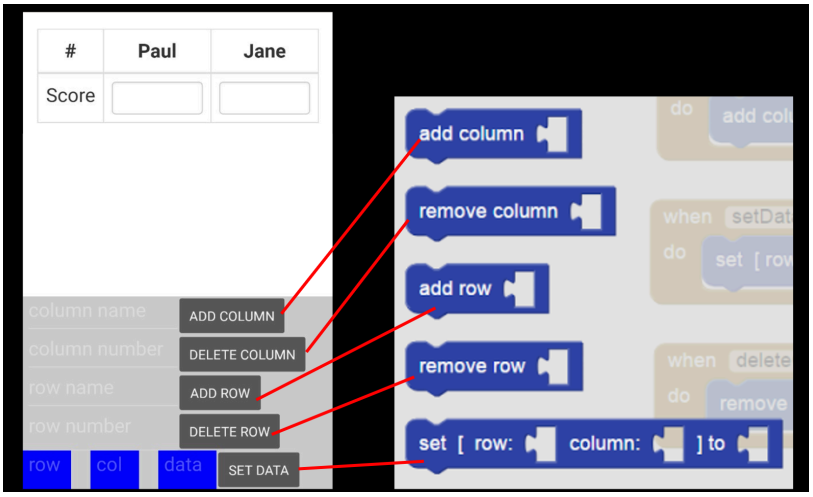

Figure 11. Corresponding blocks and buttons in app

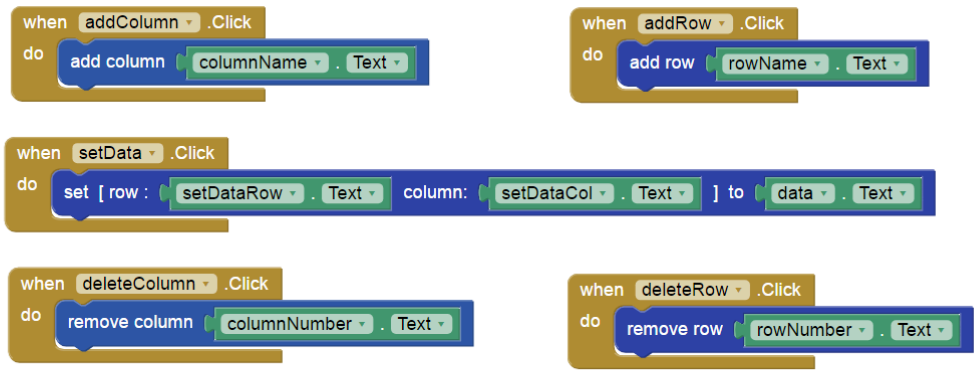

Figure 12. Blocks used to create the app in Figure 1 and Figure 11.

7. Education

I am in the process of writing a tutorial that users could follow to build an app using the table JavaScript library described in **4.Scenario**. The tutorial along with the JavaScript table API and the aia file will be available in MIT App Inventor's official website.

8. Future Work

Giving users easy access on how to make user-defined blocks, such as organizing tutorials and demos, will be an important work in the future to lower the barrier of using user-defined blocks. Enabling users to share their user-defined blocks with other users, which means sharing the JavaScript API and JSON formatted string that they used when building an app, will also be important to help out users that cannot easily find a JavaScript API to start with.

Another important area is using user-defined blocks to embed new blocks in MIT App Inventor, since user-defined blocks has the potential to extend the range of blocks App Inventor has. Therefore, ways of combining necessary JavaScript APIs with user-defined blocks should be explored, such as embedding virtual reality blocks with it.