

# **Flow BP**

Software Engineering Project

## **User Manual**

Group 4

Tomer Bitran  
Shir Markovits  
Shahar Hazan

## Contents

<b>Opening screen .....</b>	<b>3</b>
<b>Legend .....</b>	<b>3</b>
<b>Creating “Hello World” flow.....</b>	<b>6</b>
<b>Creating “Generic Hot Cold” flow .....</b>	<b>8</b>

## Opening screen



## Legend

1. Console: opens a console bar in the left side of the screen showing the events where running after running the “Run”/”Debug” (#2/#3) buttons
2. Run: after creating a flow, runs the BP program derived from the created flow.
3. Debug: after creating a flow, turns the program into “debug” mode, disables all other buttons except “stop”, “step back” and “step forward” (#4, #5, #6).
4. Stop: turn off the debug mode. Enabled only while in debug mode.
5. Step back: not implemented.
6. Step forward: enabled only while in debug mode. Show the step just happened in the created flow: every first click paints in green the selected nodes that run, and every second click paints in dark gray the active nodes that are possible to be selected in the next click.
7. Trash: deletes the flow already created and remains a clean working sheet.
8. Menu: enables the user to load a ready-made flow and save the current flow.
9. Clear: clears the presented output on the console bar.
10. The nodes menu:
  - a. Start: a node presents a new BThread. The user can initialize the payload will be transferred to the next node.

- b. BSync: a node that presents a bsync object. The user can assign an even the node will be waiting for, an event the node will block and an event to be happen when the node will be selected.

The diagram shows a node configuration window titled "2. Bsync". It contains four input fields: "request:" with the value "Hello", "wait:", "block:", and "payload:". The "payload:" field is highlighted with a dark background.

- i. Those fields can get:
1. a **string** like: "Hello"
  2. a **json** object like: {name: <event name> data: js-type}
  3. a **list of jsons** like: [ {name: <event name> data: js-type}, {name: <event name> data: js-type}]
- c. General: a node type can be placed between two nodes. This node can contain a javascript code and several outputs.

The diagram shows a node configuration window titled "7. General". It contains a "payload:" field and two output fields labeled "output1" and "output2". Below the "payload:" field are two buttons: "Edit code" and "Edit node".

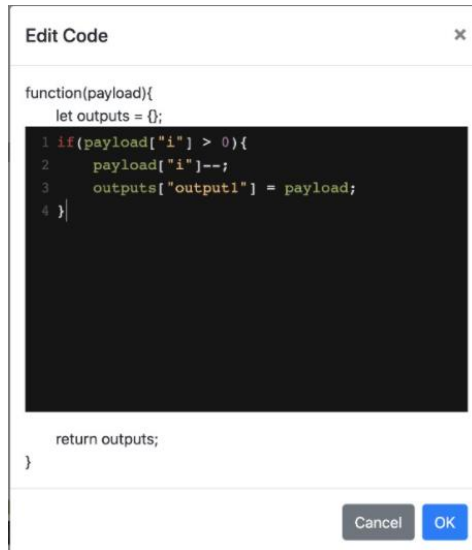
- i. Editing a General node:
1. The user can modify the node's title, the number of outputs it will have and their names.

The screenshot shows the "Edit Node" dialog box. It has a "Title:" field with the value "General" and a green checkmark. Below it is a "Number Of Outputs:" field with a value of "1" and minus/plus buttons. Underneath is a list of output names, with "output1" visible. At the bottom are "Cancel" and "OK" buttons.

ii. Editing General node's code:

1. **Note** – in the end of the javascript code the user must define the payload to pass for each output assign to the node. It should be written this way:

Outputs[<output name>] = <payload to pass>



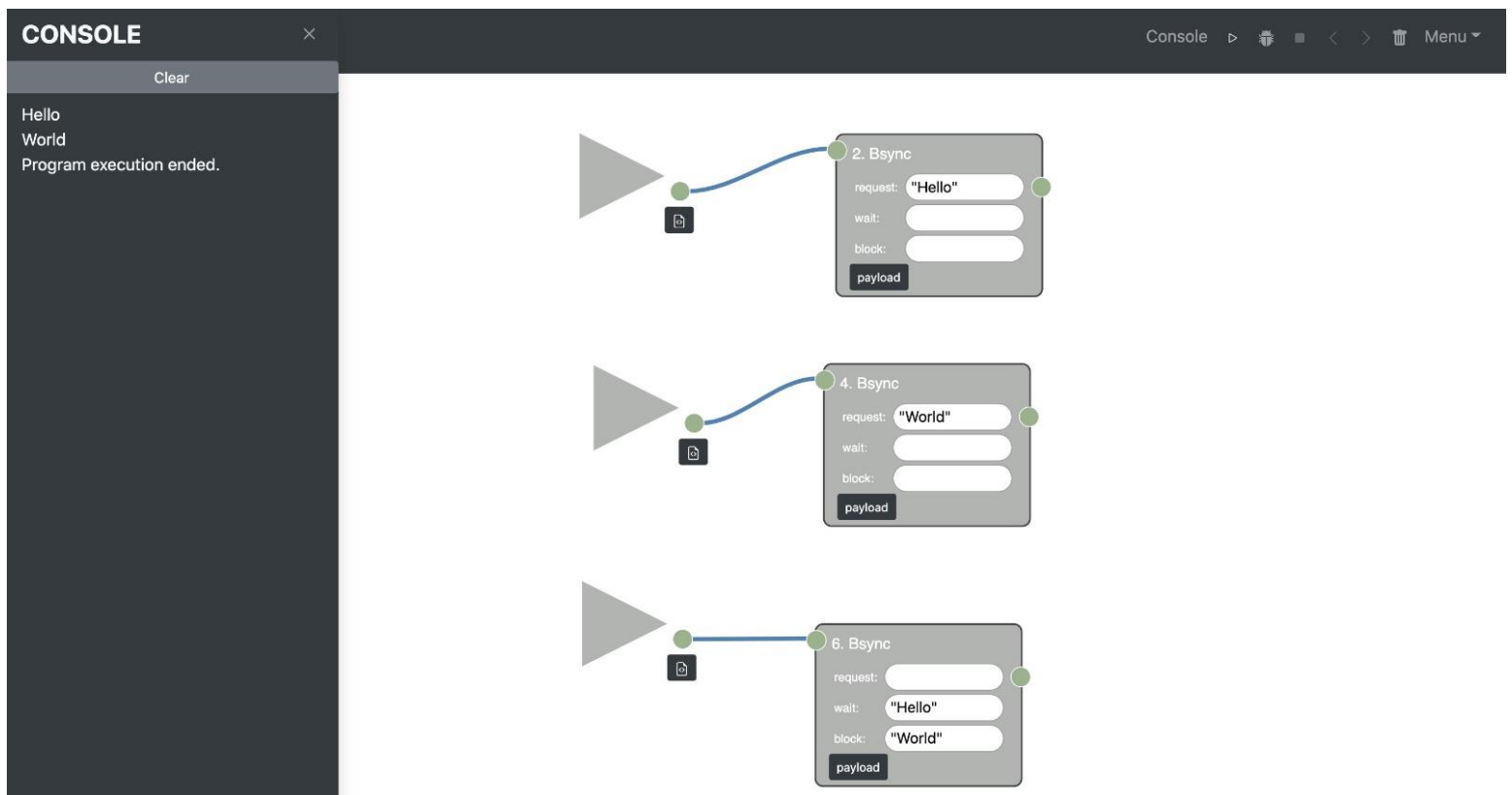
The screenshot shows a dialog box titled "Edit Code" with a close button (X) in the top right corner. The dialog contains a text area with the following JavaScript code:

```
function(payload){  
  let outputs = {};  
  
  1 if(payload["i"] > 0){  
  2   payload["i"]--;  
  3   outputs["output1"] = payload;  
  4 }  
  
  return outputs;  
}
```

At the bottom right of the dialog, there are two buttons: "Cancel" and "OK".

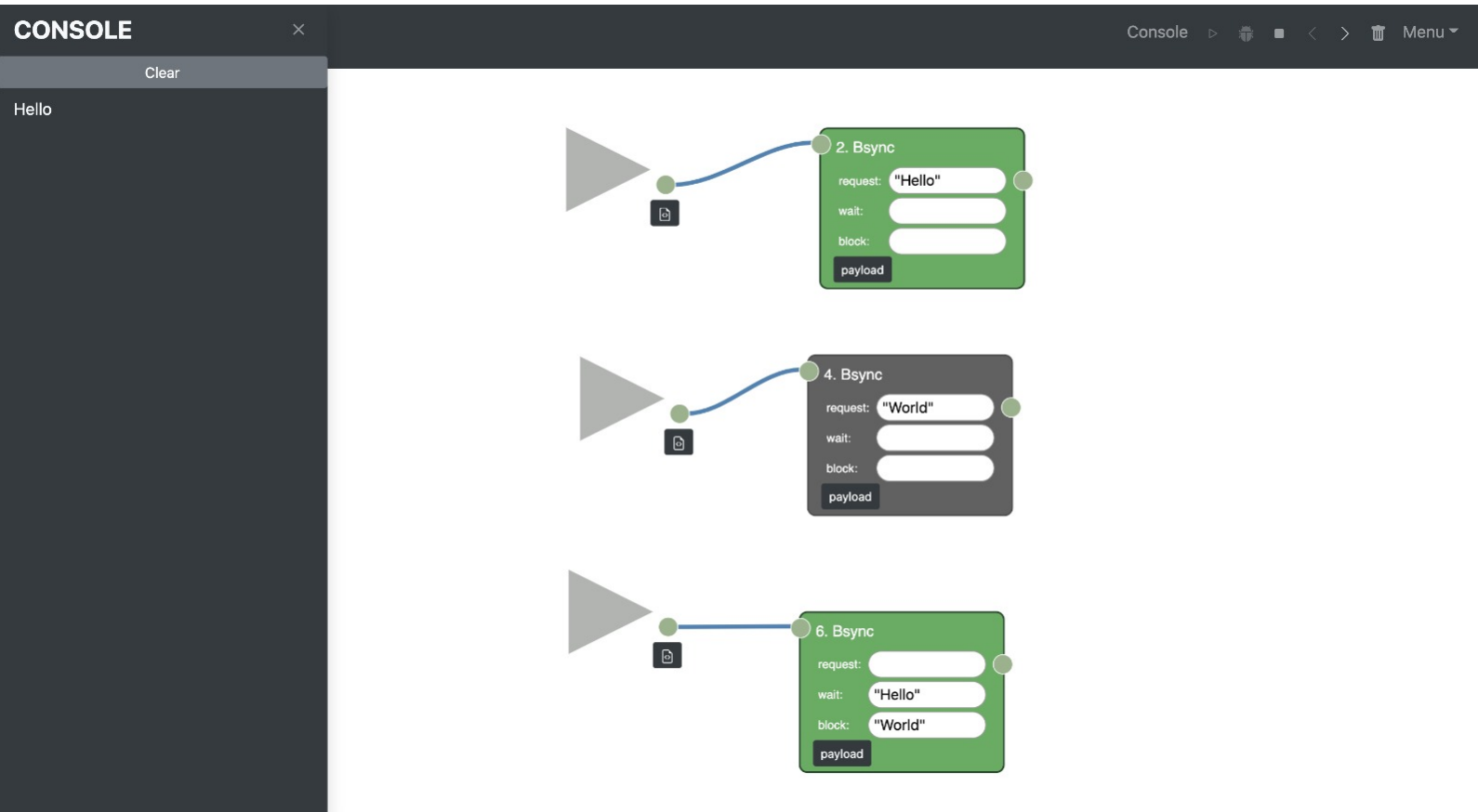
## Creating “Hello World” flow

1. Right click on the screen presents the nodes menu.
2. Create a start node to be the initializer of “Hello” event.
3. Create another start node to be the initializer for the “World” event.
4. Create another start node to be the initializer for the “arbiter” thread.
5. Create a bsync node to handle the “Hello” event.
6. Create a bsync node to handle the “World” event.
7. Create a bsync node to be responsible for waiting for “Hello” and blocking “World”.
8. Connect each start node to its relevant scenario.
9. For presenting the output of the created program click on “Console” button.
10. Click run: the events will be shown on the Console bar.



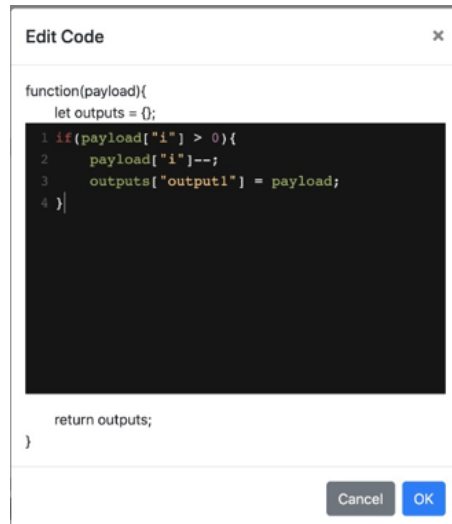
11. For debugging:

- Click “Debug” to turn into debug mode. The possible nodes to be selected will be shown in dark gray.
- Click “step forward”: the selected nodes will be shown in green.
- To turn off the debug mode click stop, to continue the debugging keep clicking “step forward” until program will be ended.



## Creating “Generic Hot Cold” flow

1. Create a start node to be the initializer of both “Hot” and “Cold” events.
  - a. Initialize the payload it passes to the connected events this way: `{“i” : 3}`  
(in this case we want each event, “Hot” and “Cold” to happen three times.
2. Create another start node to be the initializer for the “arbiter” thread.
3. Create a bsync node to handle the “Hot” event.
4. Create a bsync node to handle the “Cold” event.
5. Create two general nodes (from the two declared above) to handle the code that will make sure each one (“Hot”, “Cold”) will happen exactly three times. (in this case each node will decrement by 1 the counter each time the related event is happening).



```
function(payload){
  let outputs = {};
  1 if(payload["i"] > 0){
  2   payload["i"]--;
  3   outputs["output1"] = payload;
  4 }
  return outputs;
}
```

6. Add two bsync nodes to be the arbiters that make sure “Hot” and “Cold” will happen alternately -> first bsync node waits for “Hot” and blocks “Cold” and the second one will wait for “Cold” and will block “Hot”.
7. Connect each start node to its relevant scenario.
8. For presenting the output of the created program click on “Console” button.



9. Click run: the events will be shown on the Console bar.

CONSOLE

Clear

Hot  
Cold  
Hot  
Cold  
Hot  
Cold  
Program execution ended.

Console

Menu

The diagram illustrates a workflow with the following nodes and connections:

- Node 3. General**: Has an **output1** field. It is connected to **Node 2. Bsync** and **Node 4. General**.
- Node 2. Bsync**: Has fields for **request** (set to "Hot"), **wait**, and **block**. It is connected to **Node 3. General** and **Node 5. Bsync**.
- Node 4. General**: Has an **output1** field. It is connected to **Node 3. General** and **Node 5. Bsync**.
- Node 5. Bsync**: Has fields for **request** (set to "Cold"), **wait**, and **block**. It is connected to **Node 2. Bsync** and **Node 7. Bsync**.
- Node 7. Bsync**: Has fields for **request**, **wait** (set to "Hot"), and **block** (set to "Cold"). It is connected to **Node 5. Bsync** and **Node 8. Bsync**.
- Node 8. Bsync**: Has fields for **request**, **wait** (set to "Cold"), and **block** (set to "Hot"). It is connected to **Node 7. Bsync**.

The workflow starts with a trigger (triangle) leading to **Node 3. General**. From there, it branches into two parallel paths: one through **Node 2. Bsync** and another through **Node 4. General**. Both paths converge at **Node 5. Bsync**, which then leads to **Node 7. Bsync**, and finally to **Node 8. Bsync**. There are also feedback loops from **Node 2. Bsync** and **Node 5. Bsync** back to **Node 3. General**.

10. For debugging:
  - a. Click “Debug” to turn into debug mode. The possible nodes to be selected will be shown in dark gray.
  - b. Click “step forward”: the selected nodes will be shown in green.
  - c. To turn off the debug mode click stop, to continue the debugging keep clicking “step forward” until program will be ended.

