**Flow BP**

Software Engineering Project

Maintenance Guide

Tomer Bitran

Shir Markovits

Shahar Hazan

Introduction

[**Chapter 1 - Debug Maintenance** 3](#_Toc73889044)

[**Data flow:** 3](#_Toc73889045)

[**The implementation behind presenting the active nodes:** 3](#_Toc73889046)

[**The implementation behind presenting the selected nodes:** 4](#_Toc73889047)

[**The implementation behind presenting the nodes’ payload:** 4](#_Toc73889048)

[**Adding Blocked nodes in debug mode:** 4](#_Toc73889049)

[**Change color of specific nodes group:** 5](#_Toc73889050)

[**Chapter 2 - Run Maintenance** 6](#_Toc73889051)

[**Data flow:** 6](#_Toc73889052)

[**Chapter 3 – Extend the project** 7](#_Toc73889053)

[**Add new node type:** 7](#_Toc73889054)

[**Adding external events:** 7](#_Toc73889055)

[**Adding rete plugin:** 8](#_Toc73889056)

[**Adding new route in the server:** 8](#_Toc73889057)

[**Adding new event handler in the client:** 8](#_Toc73889058)

# **Chapter 1 - Debug Maintenance**

## **Data flow:**

Route = “/debug”:

Server:

* Sends the list of the following nodes groups:
  + active nodes
  + blocked nodes
  + nodes’ payloads
* Receives a Json represents the parsed graph and convert in to GraphModel via spring.

Client:

* Sends a Json represents the parsed graph.
* Receives the list of the following nodes groups:
  + active nodes
  + blocked nodes
  + nodes’ payloads

Route = “/step”:

Server:

* Sends:

in DebugRunner class -> step() we send the list of the following nodes groups:

* + SelectedNodes
  + active nodes
  + blocked nodes
  + nodes’ payloads
* Receives graph Id

Client:

* Sends graph Id.
* Receives nodes lists and change the selected nodes and the active nodes color using the corresponding handlers.

## **The implementation behind presenting the active nodes:**

Client:

* In Parser.js file generateBsyncCode method the code of the bsync node is generated and include insertion and removal of the current node into “active” list before and after the bp.sync code.
* Painting in dark gray during debug mode happens in colorSecondStep method.

Server:

* We initialize the active nodes list in ServiceImpl class using bprog.putInGlobalScope .
* We assign this list into “nodeLists” struct in rungraph.js.

## **The implementation behind presenting the selected nodes:**

Client:

* In Parser.js file generateBsyncCode method the code of the bsync node is generated and include insertion of the current node into “selectedNodes” after the bp.sync code.
* Painting in green during debug mode happens in stepEventHandler method.

Server:

* We initialize the selected events list in ServiceImpl class using bprog.putInGlobalScope .
* We send this list via “selectedEvents” route and handle it using selectedEventsHandler.

## **The implementation behind presenting the nodes’ payload:**

Client:

* We present the payload of each node using updatePayloads method in eventHandlers file.

Server:

* We have payload struct in rungraph.js which is a map from nodeId -> currentPayload.
* In the execution functions of each node in rungraph.js (runInNewBT, runInSameBT) we update the payloads struct.

## **Adding Blocked nodes in debug mode:**

Client:

* The stepEventHandler function in eventHandlers file handles the red painting of the blocked nodes list (nothing else needed).

Server:

* Need to maintenance the blocked nodes list in rungraph file:
  + Decide when and where a new node should be added to this list.
  + Decide when and where a new node should be removed from this list.

## **Change color of specific nodes group:**

* In BsyncComponent.js we have a map from colorName -> css class.
* In Rete.vue we have the definition of each css class.

# **Chapter 2 - Run Maintenance**

## **Data flow:**

Route = “/run”:

Server:

* Sends the events to the client in the GraphProgramRunnerListener class.
* Receives a Json represents the parsed graph and convert it to GraphModel via spring.

Client:

* Sends a Json represents the parsed graph.
* Receives the selected events via the event listener of "flowEvent"(event handler is added in the init function in index.js).

# **Chapter 3 – Extend the project**

## **Add new node type:**

**All the following steps are done in the client.**

1. Create new file in src->node-editor->components folder.
2. Create class which extends AbstractComponent with the following functions:
   1. Constructor – call super(componentName)
   2. Builder(node) –
      1. the first line should be:

node = AbstractComponent.prototype.builder(node,  <numOfOutputs>,<OutputTitlesList>);

* + 1. Define the node default code by assign node.data.code.
    2. Add controls to the node by using node.addControl(<Control>), Control can be one of the following:
       1. InputTextControl – constructor take name as arg.
       2. CodeControl – constructor take node outputs and node id.
       3. PayloadControl – constructor take nodeData and node id.
    3. In the end of the function return node.

1. In index.js(the init file):
   1. Add your new node component to the components list.

**\*\*ForComponent is a good example for how to add new node\*\***

## **Adding external events:**

Client:

* adding a new node which represents external event (see below how adding a new node-type).
* In ParseNode function in Parser.js file give a unique type to the new node.
* Add new button in Rete.vue indicates a program with external event is about to run.

Server:

* Add new route in Controller class to handle a program with external events.
* Add new function in ServiceImpl class, the function should be like run function with the following changes:
  + It should turn on the flag of BProgram for external events by the method setWaitForExternalEvents of BProgram.
  + For each node of the new type create an event and use enqueueExternalEvent method of BProgram.

## **Adding rete plugin:**

There are few plugins of rete that described here:

<https://rete.js.org/#/docs/plugins/connection>

To add a plugin following the next steps in the client:

1. Add the name and version of the plugin in the "dependencies" section in "package.json".
2. Run npm intall.
3. In index.js:
   1. Import the plugin.
   2. In the init function, add:
      1. Editor.use(<plugin>);

## **Adding new route in the server:**

Add function in Controller class by the following format:

@PostMapping(value = "/<route name>", consumes = "application/json", produces = "application/json")  
public <return type> <route name>(@RequestBody <Param type> <Param name>){  
 <code…>  
}

Then you can use the function post in controller file in the client to trigger the new route.

## **Adding new event handler in the client:**

1. Create new event handler function(that take event as param) in EventHandler.js.
2. In index.js -> init function:
   1. Use eventSource.addEventListener(<event name>, handler from 1.).

For trigger this event handler from the server you should do:

1. Use the SseEmitter of the client and use it by the following format:

emitter.send(SseEmitter.*event*().name(<event name>).data(<dateToSend>));