Application for Flow Manipulation API

8/30/2018

Kunihiko Toumura

0. Summary

- We currently implementing Flow Manipulation API as an independent NPM package.

- In order to implement application using Flow Manipulation API, we are investigating how to extend Node-RED functionality in a pluggable manner.
  - Methods for extend functionality by NPM module.
    - Node-RED node module mechanism can be used.
  - Methods for extend UI (menu items, buttons) that invoke extended function.
    - RED.menu.addItem() can be used, but no API for buttons.
    - Q: Is there any plan for create API to add button on UI?
  - Methods for extend Web API endpoint that handles extended function.
    - RED.httpAdmin.{get, post,…}() can be used.
    - Q: Is there any convention or rule for endpoint pathname?
  - Methods for add hook for Admin API (PUT /flows etc.)
    - Currently, there is no API for add hook.
    - Q: When Editor-Runtime API is implemented, I think it will be implemented as custom handler for RED.flows.setFlows(). Is this right?
- Q: If it is necessary to add new API on Node-RED core, is there suitable timing for incorporate it (e.g. on 0.20)?

2. Application for Flow Manipulation API

We are planning to publish the API with a concrete application

Objective:
- To verify whether the API set is enough to implement an apps.
- To investigate how to implement these apps as a pluggable manner.

Requirements:
- The implementation requires a minimum change in current Node-RED. If needed, we should also design clear API for these functions.

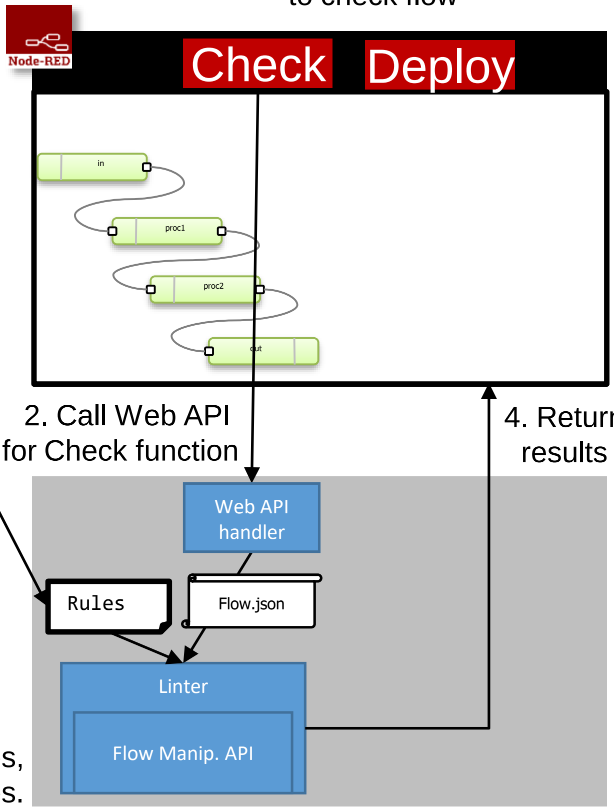
3. An Application Idea --- Linting tool for flows

Check flow whether it adhere to certain style guidelines.

Implementation Pattern 1: Add extra UI for call function

Node-RED editor

1. Push check button or select menu to check flow



2. Call Web API for Check function

4. Return results

3. Receive flow and rules, And check it adhere to rules.

server

Rules for flow

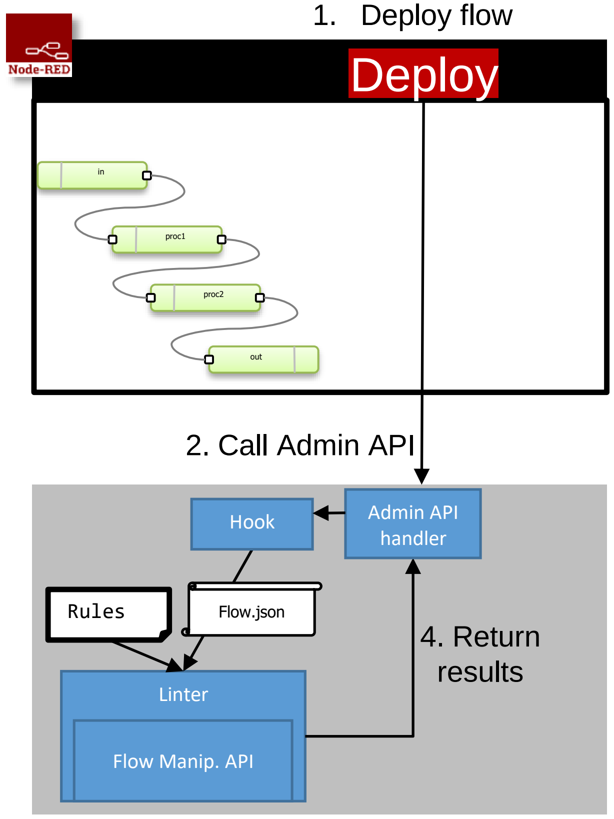
```
Rules for JS code in function node:
  indent: 2
  require semicolons

Rules for Node
  max node per tab: 100
  max link length: 2000
Etc.
```

(in setting.js or Project settings?)

Implementation Pattern 2: Hook Admin API call

1. Deploy flow



2. Call Admin API

4. Return results

4. Extentions

To plug a lint function, following extension is needed

Implementation Pattern 1: Add extra UI for call function

1. Add Menu item or button to invoke the function.
2. Write the function as ‘onselect’ or ‘click’ handler.

-> Requirements:
- Extend functionality by npm module:
  - Node-RED node module mechanism can be used.
- API for adding UI:
  - menu item: RED.menu.addItem() can be used.
  - button: not exist. Need to write code in <script/>.
- API for adding Web API handler for the function:
  - RED.httpAdmin.{get, post,…}() can be used.

Implementation Pattern 2: Hook Admin API call

1. Add hook for Admin API (POST /flows, POST /flow, PUT /flow/:id)
2. By the hook, custom code for flow manipulation is called.

-> Requirements:
- API for hook call of admin API

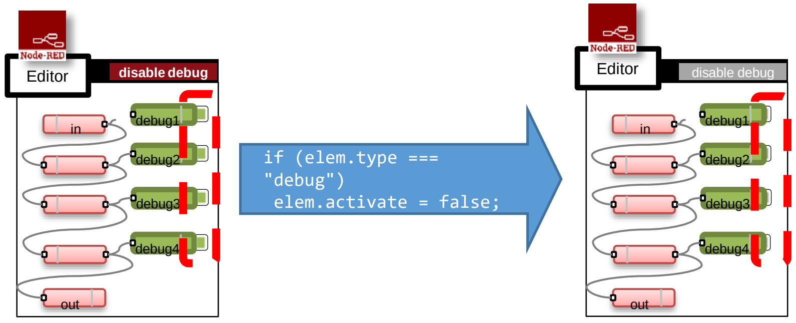
Supplemental material

A.1 Use Case of Flow Manipulation API

HITACHI
Inspire the Next

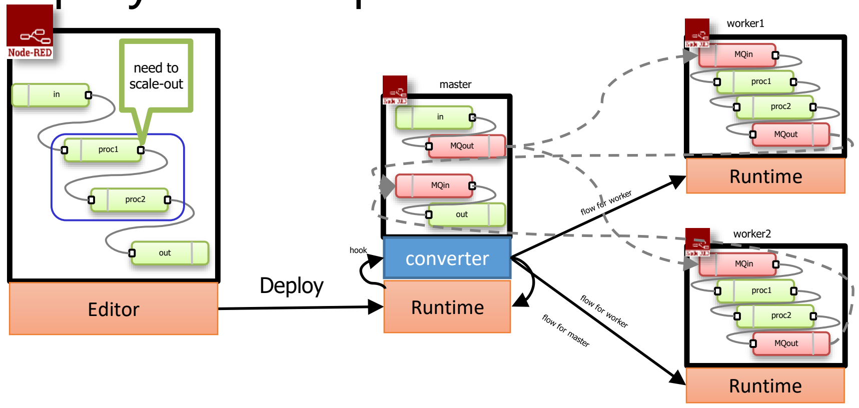
Extend functionality of Node-RED flow editor / runtime
via unified interface for translating or modifying a flow definition.

Use case 1: UC1
Inactivate all debug node in one click

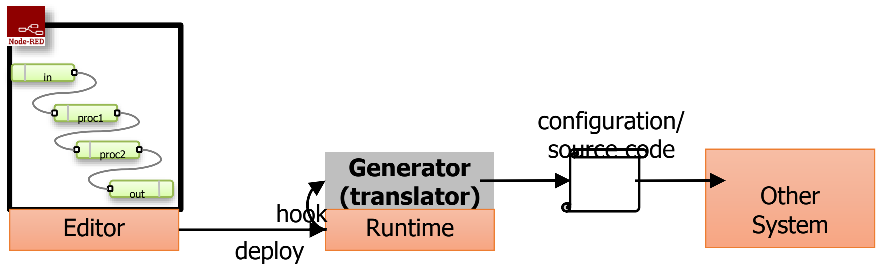


```
if (elem.type ===
"debug")
  elem.activate = false;
```

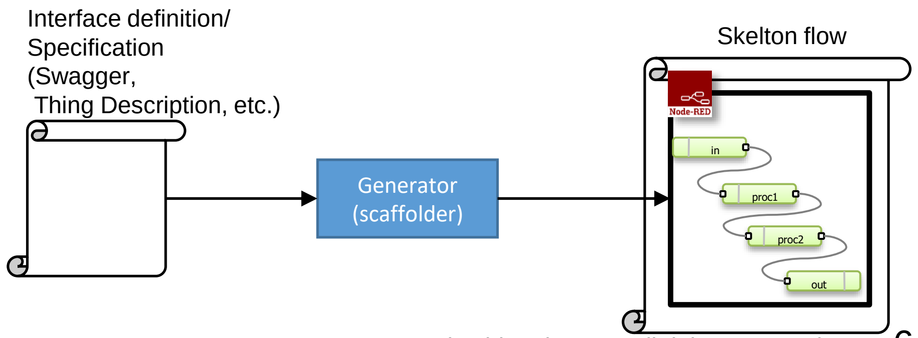
Use case 2: UC2
Deploy to multiple Node-RED runtime



Use case 3: UC3
Use Editor in Other Systems



Use case 4: UC4
Generate flow from Other Configuration/Definition Language



A.2 Current usage of API path in core-module

httpAdmin

Debug node:
- POST /debug/:id/:state
- GET /debug/view/*

Inject node:
- POST /inject/:id

Rpi-GPIO node:
- GET /rpi-gpio/:id
- GET /rpi-pins/:id

UDP node:
- GET /udp-ports/:id

httpNode

httpin node:
- ANY httpNodeRoute+path

Websocket node:
- httpNodeRoute+path