HITACHI
Inspire the Next

Flow Linter

27 January 2020

Kunihiko Toumura

Research and Development Group
Hitachi, Ltd.

© Hitachi, Ltd. 2020. All rights reserved.

HITACHI
Inspire the Next

Contents

1. Background, and Use case of Flow Linter
2. Current Development status and issues
3. Work items for this week

© Hitachi, Ltd. 2020. All rights reserved.

1-1. Introduction

HITACHI
Inspire the Next

- Development of Node-RED flow has high flexibility, but these flexibility cause the lack of consistency of coding style, or even worse, make a development error-prone.
- Other language has validation tools:
  - JavaScript: ESLint
  - Java: CheckStyle, FindBugs
  - C: lint
  - Python: Pylint, etc…

-> We propose a validation tool for Node-RED flow.

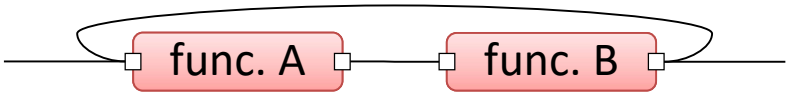
© Hitachi, Ltd. 2020. All rights reserved.

1-2. Use case of Flow Linter (1/2)

HITACHI
Inspire the Next

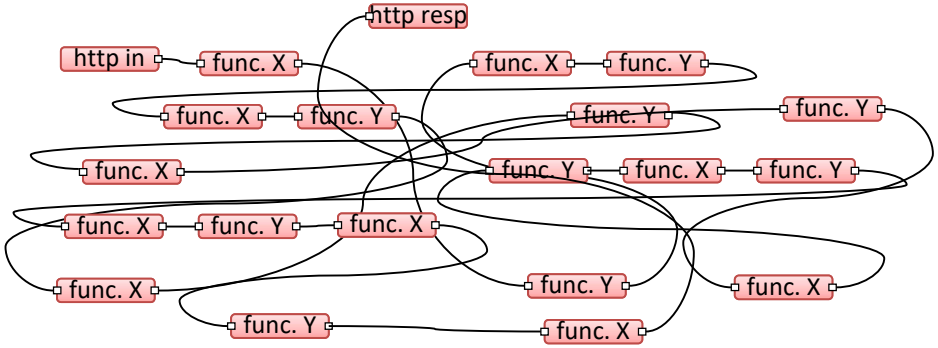
Find a potential problems in flows

- infinite loops

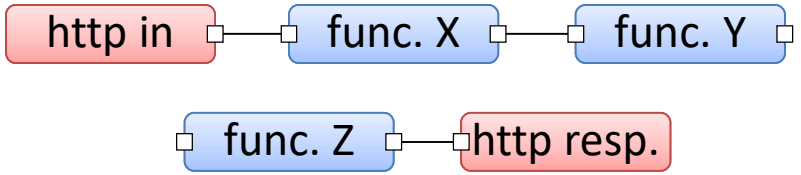


- function node without name

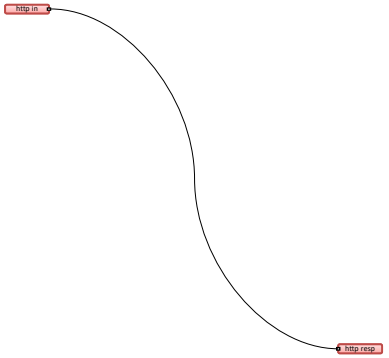
- excessive amount of nodes in a flow



- unmatching HTTP-in/response pair



- excessive size of a flow

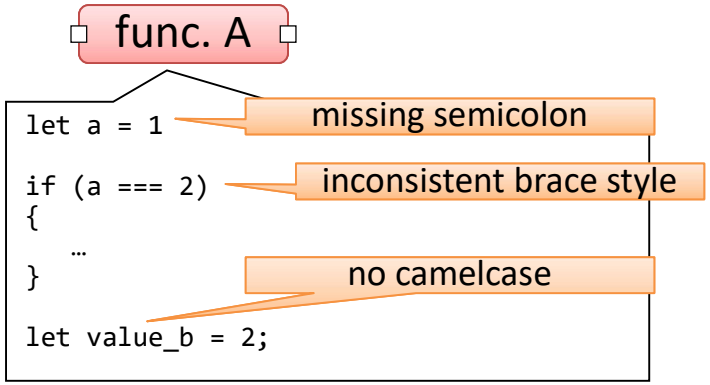


© Hitachi, Ltd. 2020. All rights reserved.

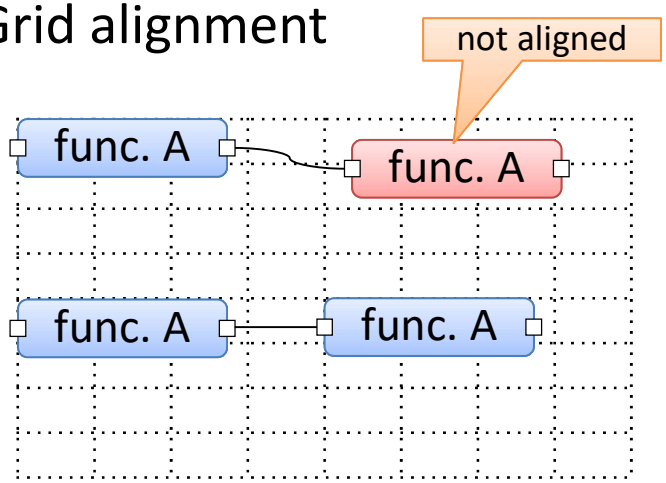
HITACHI
Inspire the Next

Align with coding style

- Coding style in Function node



- Grid alignment



Because rules may vary among flow developers,
rules should be extendable by them.

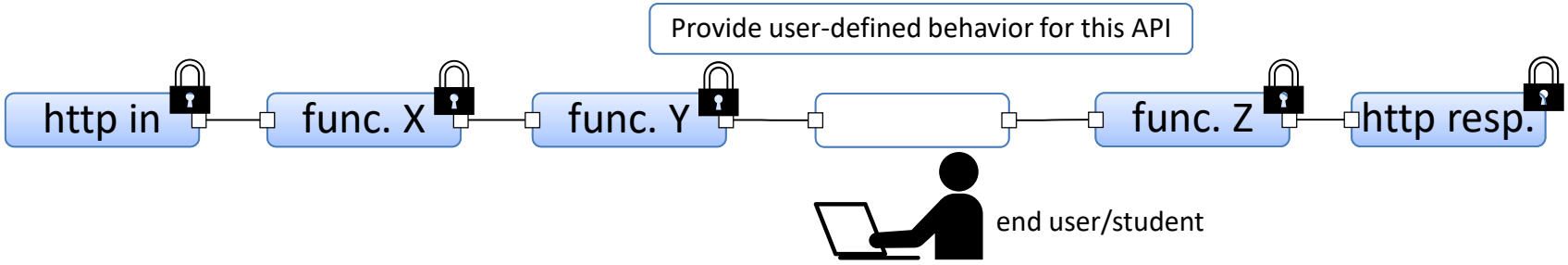
© Hitachi, Ltd. 2020. All rights reserved.

1-4. Advanced use cases of editor-linter integration

HITACHI
Inspire the Next

- Template for end-user customization, or programming education

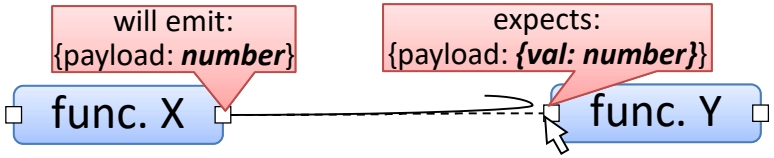
By restricting user actions for a part of flow, template provider safely avoids an accidental modification of internal logic of the flow. It can be used for educational purpose.



- Auto completion / correction

Based on metadata of nodes (e.g. schema of input/output message, node type), editor automatically places related node, or display suggestions.

As developer drops http-in node to editor,
http-response node is automatically placed on the flow



If data schema of output and input are not compatible,
a link resist to connect.

© Hitachi, Ltd. 2020. All rights reserved.

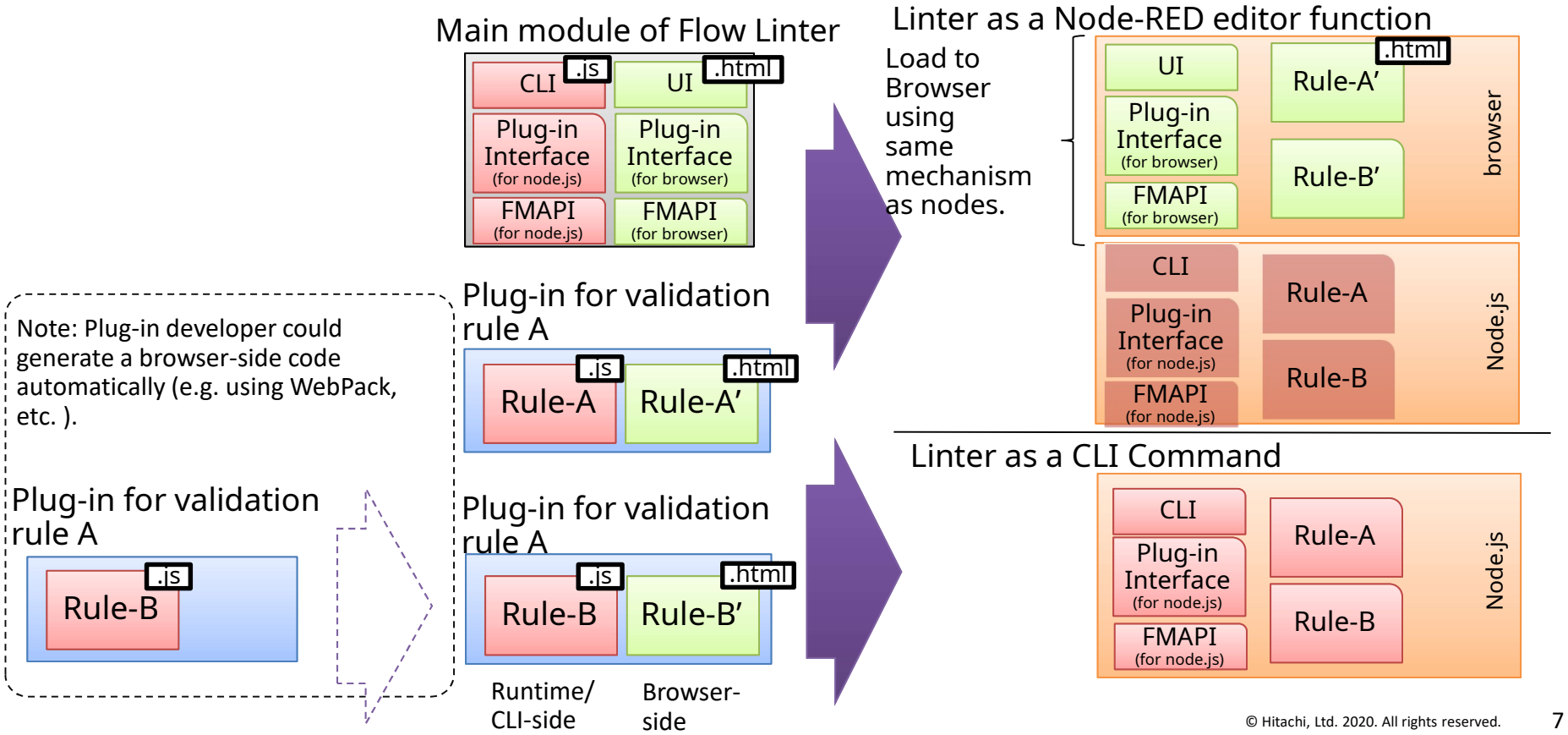
2-1. Current development status

HITACHI
Inspire the Next

- Current status:
  - Designing pluggable rule architecture
    - details are in following pages
  - Implementing CLI version of flow linter
  - Improving Flow Manipulating API and Rules
    - API: focused on designing read/search API
    - Rules: Loop detection, Empty name on Function node, Flow size, Apply ESLint for Function node.
- Next steps:
  - Design and implementation of Editor-integrated flow linter
- Related Pull Request / Repository
  - Design Note: https://github.com/node-red/designs/pull/1
  - Code: https://github.com/node-red-hitachi/node-red-flow-linter
    - When the design of flow linter is approved, I'd like to move this repository to Node-RED repository.
      - https://github.com/node-red/node-red-flow-linter etc.

© Hitachi, Ltd. 2020. All rights reserved.

2-2. Overall architecture

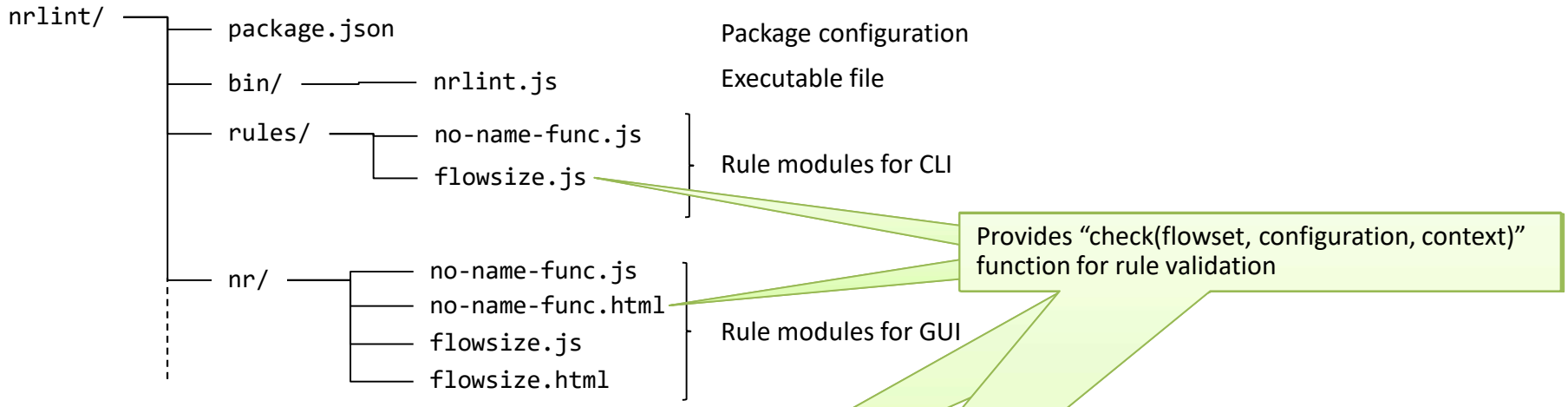


© Hitachi, Ltd. 2020. All rights reserved.

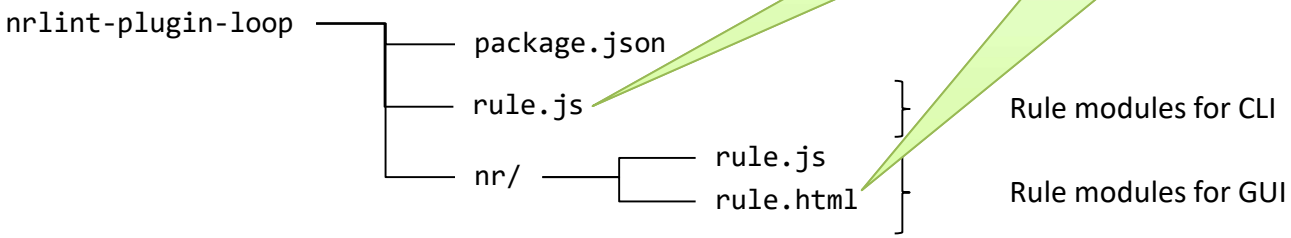
2-3. Plugin Interface

HITACHI
Inspire the Next

Main module structure



Rule plug-in module structure



© Hitachi, Ltd. 2020. All rights reserved.

2-4. Usage of command-line interface and configuration

HITACHI
Inspire the Next

Basic usage:

```
% nrlint [-c configuration-file] flow.json
```

Configuration file (default: ~/.nrlintrc.js) :

```
module.exports = {
    "rules": [
        {
            name: "no-func-name",
            mode: "warn",
        },
        {
            name: "http-in-resp"
        },
        {
            name: "loop",
        }
    ]
}
```

Name of rule plug-in

configuration of each plug-in

© Hitachi, Ltd. 2020. All rights reserved.

2-5. Output format

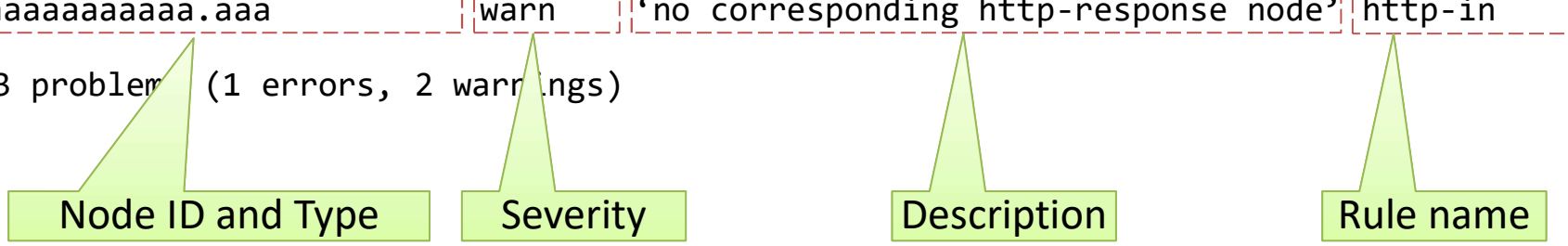
HITACHI
Inspire the Next

Similar to eslint's default (stylish) format.

```
% nrlint flow.json
/home/user/flow.json
  012345abcde.abc          error   'function node has no name'              func-no-name
  abcde012345.def          warn    'possible infinite loop'                 loop
  aaaaaaaaaaa.aaa          warn    'no corresponding http-response node'    http-in

× 3 problem  (1 errors, 2 warnings)

%
```

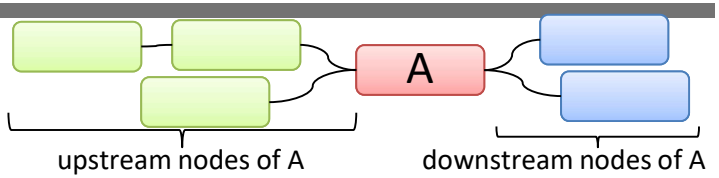


(more formats can be implemented, e.g. JSON format for machine processing, etc.)

© Hitachi, Ltd. 2020. All rights reserved.

2-6. Flow Manipulation API

HITACHI
Inspire the Next



- Added rule-plugin for check matching of HTTP-in and HTTP-response nodes
  - Add search functions to Flow Manip. API
- Plug-in code generation mechanism for browser will be considered after specification of command-line interface version is stabilized.

Current list of Flow Manipulation API (class FlowSet):

Category	function	description
create	FlowSet.**parseFlow**(*parsed flow.json*) -> *FlowSet*	create FlowSet object from flow.json file
read	FlowSet.prototype.**getAllNodesArray**() -> [FMNode]	dump all node as array.
	FlowSet.prototype.**get{Node/Flow/Config/Subflow}**(*node-id*) -> *{FMNode/FMFlow/FMConfig/FMSubflow}*	get {node/flow/config/subflow} by ID
search	FlowSet.prototype.**{next/prev}**(*node-id*) -> *[node-id]*	get nodes which are directly connected on {output/input} ports of the node.
	FlowSet.prototype.**{downstream/upstream}**(*node-id*) -> *[node-id]*	get all nodes which can be followed from the {output/input} port of the node.
	FlowSet.prototype.**connected**(*node-id*) -> *[node-id]*	get all nodes which can be followed from the output or input port of the node. (i.e. fs.downstream(n) + fs.upstream(n))

2-7. Demo (verification via command-line)

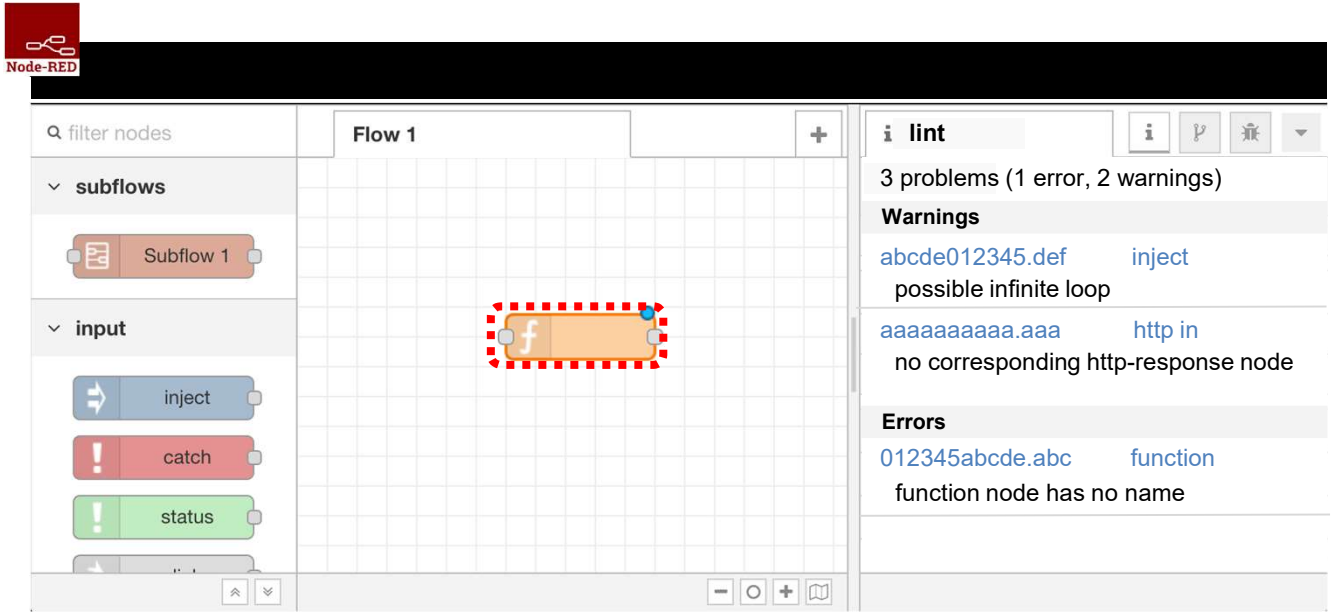
HITACHI
Inspire the Next

© Hitachi, Ltd. 2020. All rights reserved.

2-8. Integrating with Editor (to be implemented)

HITACHI
Inspire the Next

- Each time when the flow is edited, rule validation is executed.
- As same as debug side bar, problem is shown in “lint sidebar”.
  - When clicked each problem, correspondent nodes are highlighted.



© Hitachi, Ltd. 2020. All rights reserved.

2-9. Plugin implementation patterns

HITACHI
Inspire the Next

- When linter is integrated to Editor,
  where the rule validation should be executed?

	Pros	Cons
Editor	• Quick response • No inter-process communication	• Duplicate code (even if it can be automatically generated) • Need to support various browsers
Runtime	• Single code for Node.js LTS • can use a rich set of npm modules (e.g. eslint) • “Language Server Protocol[1]” uses this approach	• Any flow update causes REST API call between editor and runtime

In this phase, we are designing as “validation in Editor” approach, but automatic validation code generation is currently out of scope. (i.e. it is a plug-in developer’s duty to write the Editor-side code)

[1] https://langserver.org/

© Hitachi, Ltd. 2020. All rights reserved.

3. Work item for this week

HITACHI
Inspire the Next

CLI-version

- Discussion about new rule ideas and implementation of them.
- Brush-up Flow Manipulation API through above discussion

GUI-version

- Designing an UI

© Hitachi, Ltd. 2020. All rights reserved.