Довідник Node-RED

з елементами опису Java Script, JSON, JSONata

Про українську редакцію довідника

Довідник перекладений українською мовою і доповнений супутнім матеріалом з метою спрощення вивчення Node-RED студентам та спеціалістам з автоматизації та компютерно-інтегрованих технологій. Зокрема даний довідник використовується в курсі «Технології індустрії 4.0», що читається на кафедрі ІАСУ Національного університету харчових технологій.

- Переклад не є точним і в деяких випадках має доповнення для кращого розуміння і сприймання автоматниками. У будь якому випадку на оригінальну версію в тексті є посилання.
- Якщо після перекладеного абзацу тексту є оригінальний (англійський) його варіант, то цей абзац потребує точнішого перекладу.

Редакція 0.7.

- Добавлено опис вузлів Watson IoT
- Добавлено опис вузлів Watson IBM IoT

Задачі для наступних редакцій

- виправити чорнову редакцію розділів «Запуск Node-RED (Running Node-RED)» та «Розширення»
- заповнити розділ «Вузли – короткий довідник з посиланнями»
- змінити зовнішні посилання в тексті (окрім назв розділів) на внутрішні розділи
- доробити розділи Запуск Node-RED
- добавити розділи:
  - udp, tcp

Автори і зворотній звязок

Обговорення на http://asu.in.ua/viewtopic.php?f=290&t=2192
Перекладач, автор прикладів - Олександр Пупена pupena_san@ukr.net
https://www.facebook.com/pupena.san
У роботі над посібником також брали участь: Михайло Лапін, Оксана Лінкевич.
У наступних редакціях планується добавляти розділи з описом популярних модулів.
Долучайтеся.

Вузли – короткий довідник з посиланнями

INPUTS

Вузол	Зображення	Опис	Примітка
Inject		ручний запуску потоку, автоматичний запуску потоків через регулярні інтервали часу	Основні вузли

OUTPUTS

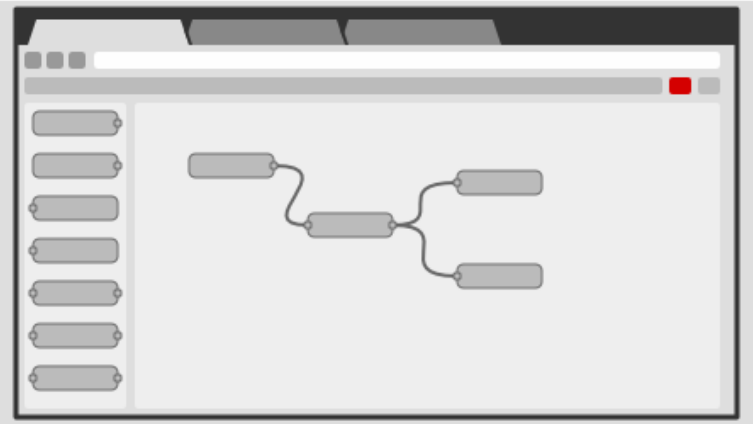
FUNCTION

Цей розділ потребує розробки.

Про NODE-RED (Node-RED)

Node-RED - це інструмент програмування для об'єднання апаратних пристроїв, API та сервісів онлайн новими цікавими способами.
Редактор Node-RED базується на браузері, який дозволяє легко об’єднувати в потоки вузли з широкого набору палітри, які можуть бути розгорнуті для виконання лише одним клацанням миші.

Редагування потоку на основі браузера

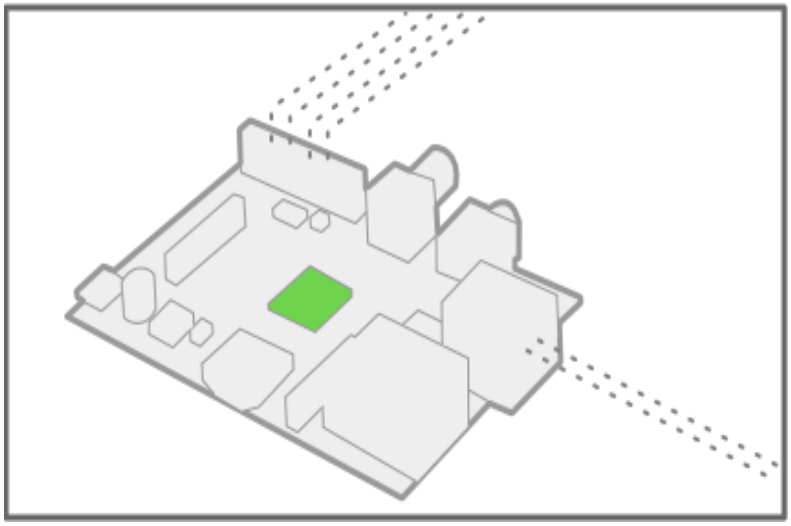


Node-RED забезпечує редактор потоку (flow editor) на основі браузера, що полегшує зв’язування потоків (flow) за допомогою широкого набору вузлів (node) палітри. Потоки можуть потім бути розгорнуті в середовище виконання в один клік.

В редакторі Node-RED за допомогою текстового редактора можуть бути створені функції на JavaScript.

Вбудована бібліотека дозволяє зберігати корисні функції, шаблони або потоки для повторного використання

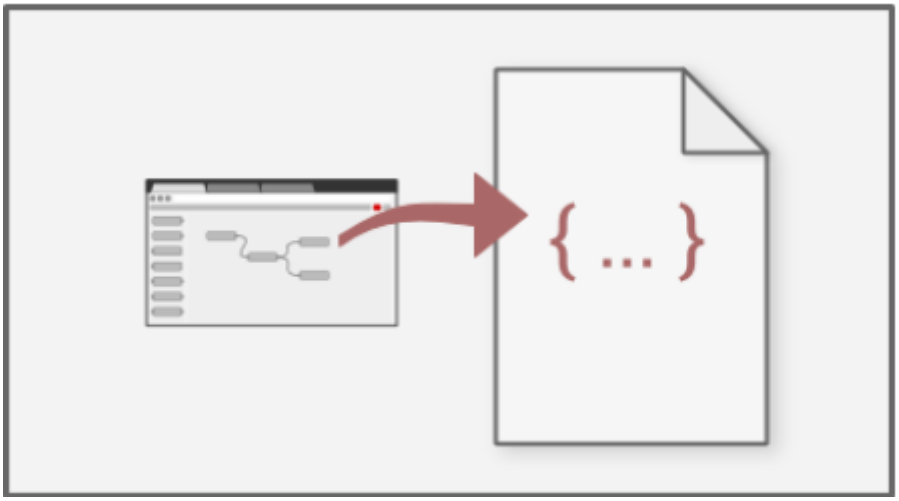
Побудовано на Node.js



Легке середовище виконання (runtime) побудоване на Node.js, користується бере що говорить про розширений набір керування подіями та не блокуючими моделями. Це робить його ідеальним для роботи на недорогих апаратних засобах, таких як Raspberry Pi, а також у хмарних сховищах.

Діапазон вузлів палітри легко збільшити новими можливостями додаванням більш ніж 225 000 модулів у сховищі Node.

Соціальний розвиток



Теки, створені в Node-RED, зберігаються за допомогою JSON, які можна легко імпортувати та експортувати для спільного використання з іншими.

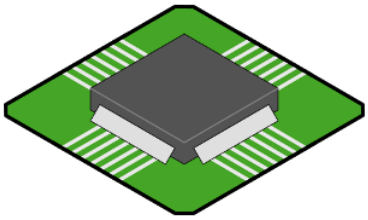
Бібліотека он-лайн потоків (flow) дозволяє вам поділитися своїми найкращими потоками з усім світом.

Початок роботи

Node-RED побудований на Node.js, використовуючи повну перевагу його подіє-орієнтованої не блокуючої моделі. Це робить його ідеальним для роботи на краю (Edge) мережі на недорогих апаратних засобах, таких як Raspberry Pi, а також у хмарі

Виконати локально

- Getting started
- Docker



На пристрої

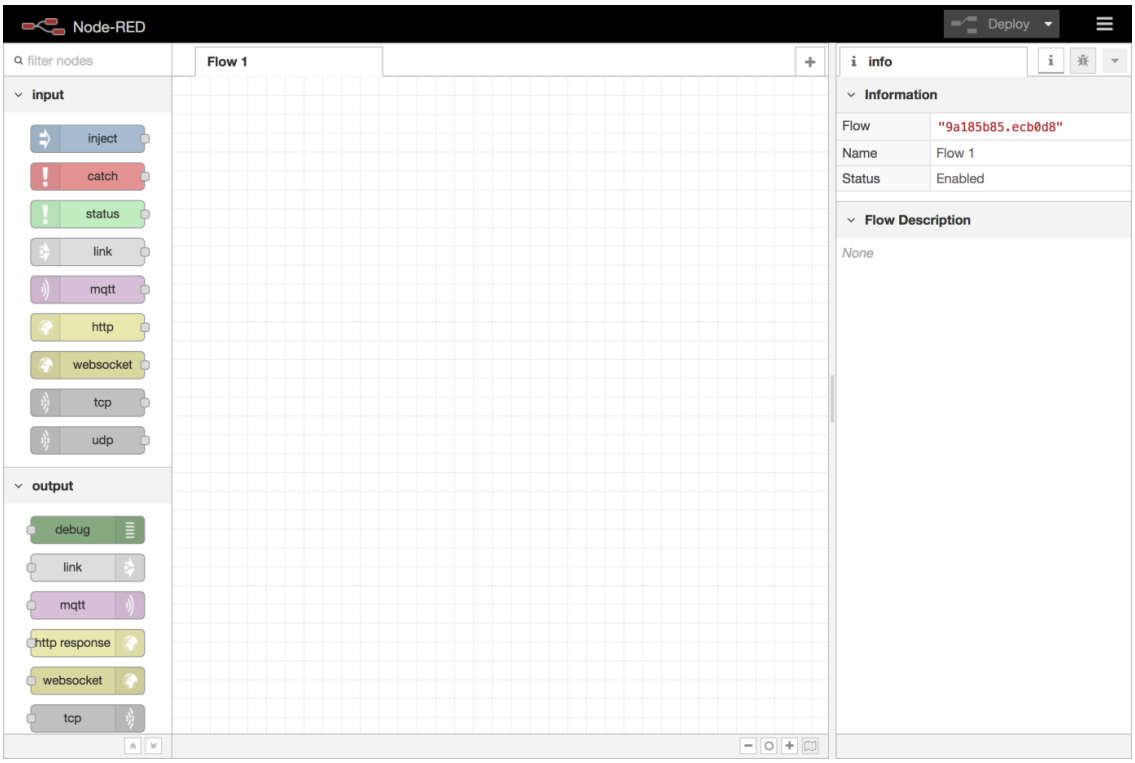
- Raspberry Pi
- BeagleBone Black
- Interacting with Arduino
- Android

У хмарі

- IBM Cloud
- SenseTecnic FRED
- Amazon Web Services
- Microsoft Azure

Посібник користувача (User Guide)

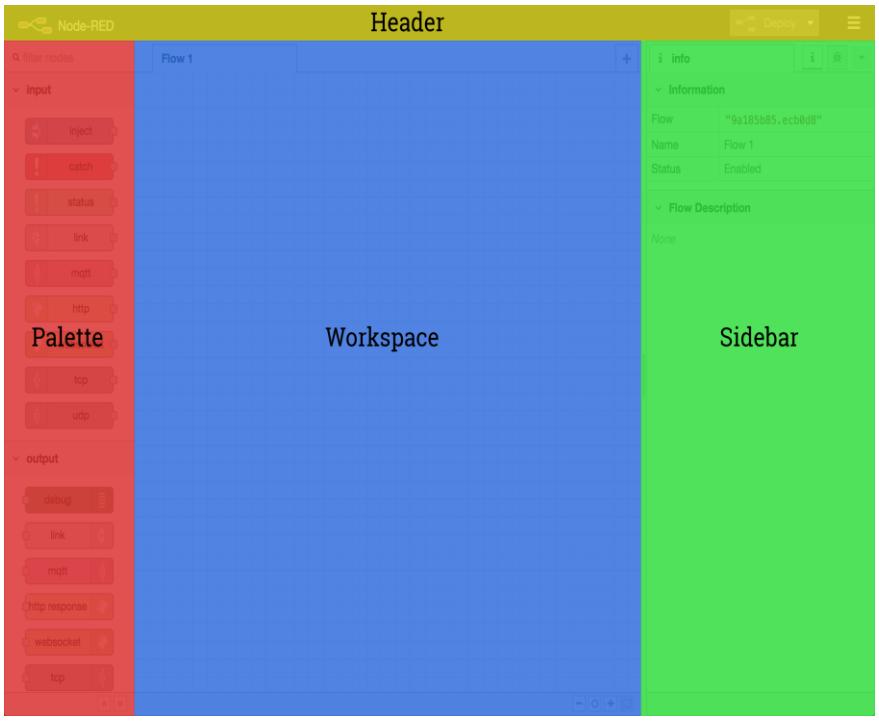
Редактор NODE-RED (Editor Guide)



Вікно редактора

Складається з таких 4 компонентів :

- У верхній частині міститься заголовок, що містить кнопку розгортання, головне меню, і якщо користувач пройшов автентифікацію, меню користувача.
- ліворуч знаходиться **палітра (palette),** яка містить вузли доступні для використання
- Посередині знаходиться основна **робоча**



область(workspace), в якій створюються потоки
- праворуч знаходиться бічна панель

Компоненти редактора

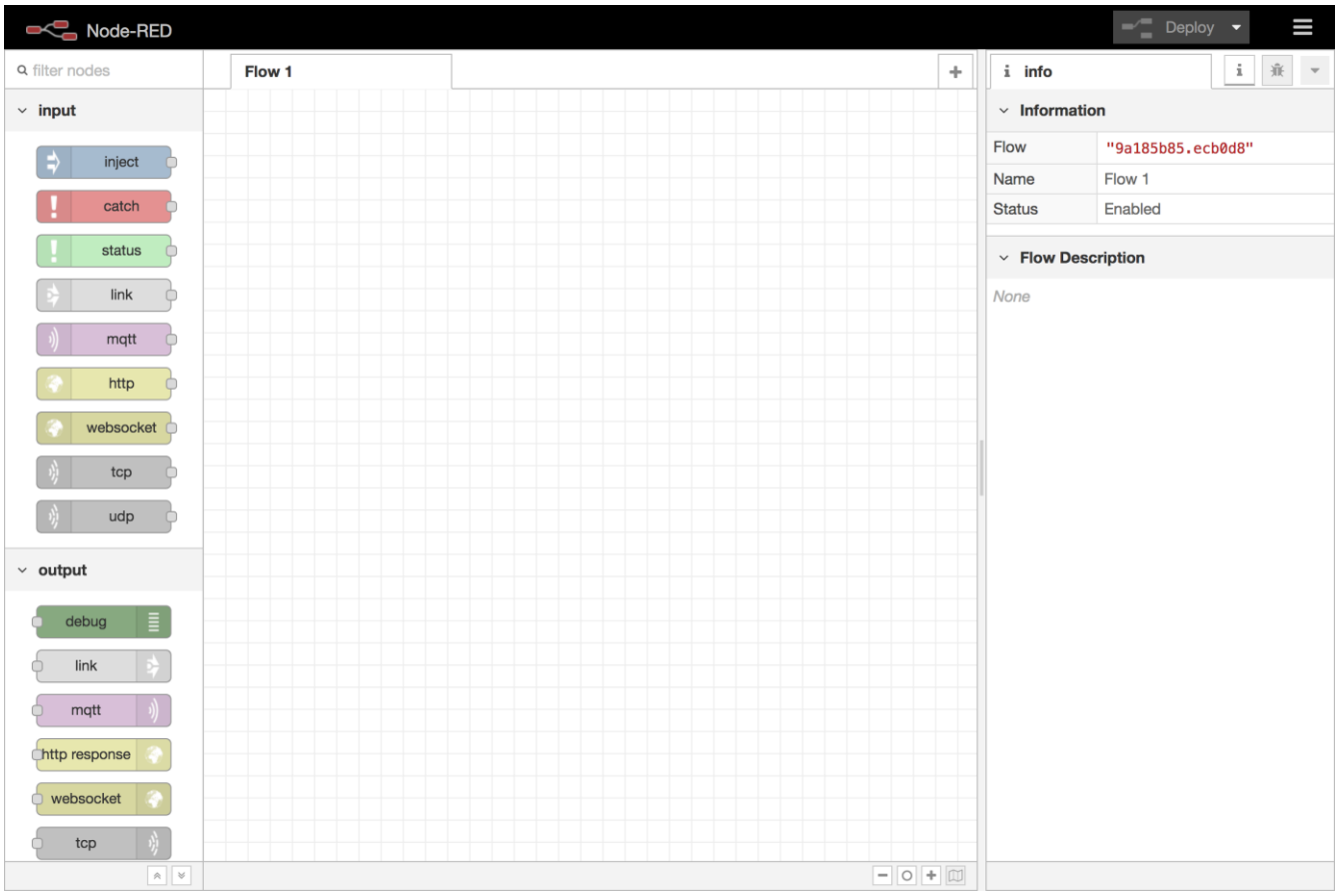
The editor window consists of four components:

- The header at the top, containing the deploy button, main menu, and, if user authentication is enabled, the user menu.
- The palette on the left, containing the nodes available to use.
- The main workspace in the middle, where flows are created.
- The sidebar on the right.

Робоча область (workspace)

Основна **робоча область** (**workspace**) - це місце, де розробляються потоки (flow) шляхом перетягування з палітри і з'єднання між собою вузлів (nodes).

Робоча область має ряд вкладок вздовж вершини - по одній для кожного потоку і будь-яких під потоків, що були відкриті.



Інструменти перегляду

Нижній колонтитул містить кнопки для збільшення/зменшення та відновлення стандартного рівня масштабування.
Він також містить кнопку перемикання для навігатора перегляду.

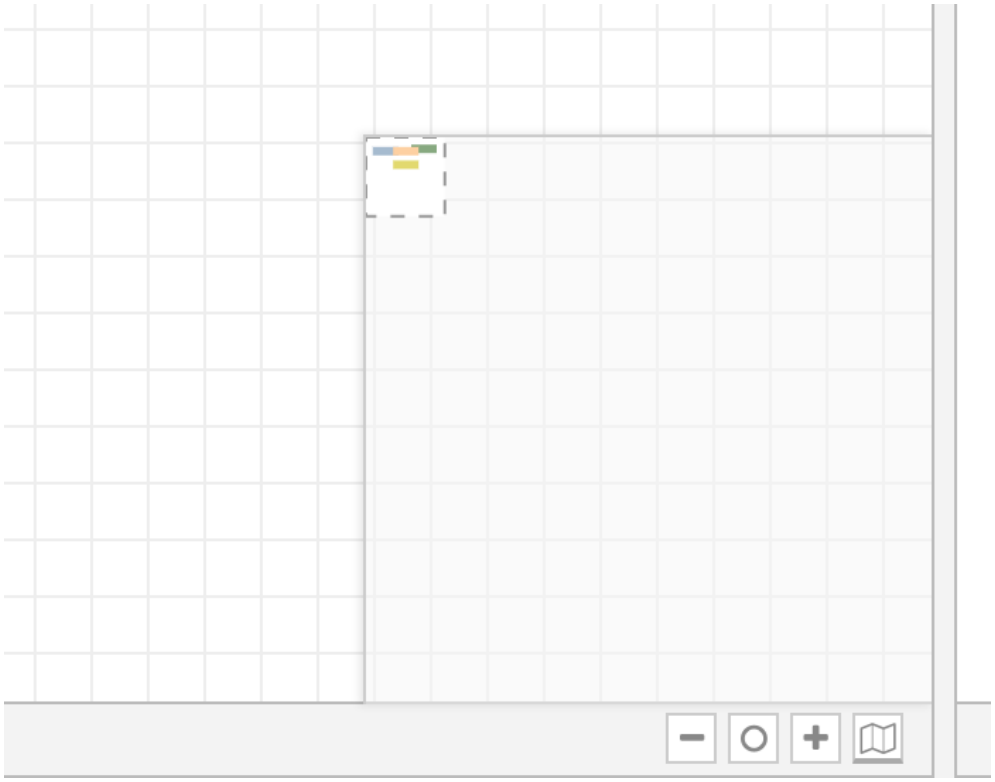


Рис. робоча область нижнього колонтитула з активним навігатором перегляду.

Reference	
Action	`core:zoom-in`
Key shortcut	`Ctrl/⌘-=`

Reference	
Action	`core:zoom-reset`
Key shortcut	`Ctrl/⌘-0`

Reference	
Action	`core:zoom-out`
Key shortcut	`Ctrl/⌘--`

Навігатор перегляду забезпечує зменшений вигляд всієї робочої області, виділяючи ту зону, яку необхідно відобразити. Цю зону можна перетягнути всередині навігатора, щоб швидко перейти до інших частин робочої області. Це також корисно для пошуку вузлів, які були "втрачені" на інших краях робочого середовища.

Reference	
Action	core:toggle-navigator
Key shortcut	*none*

Налаштування вигляду

Вигляд робочої області можна налаштувати за допомогою вкладки " View’ в діалоговому вікні " User Settings "

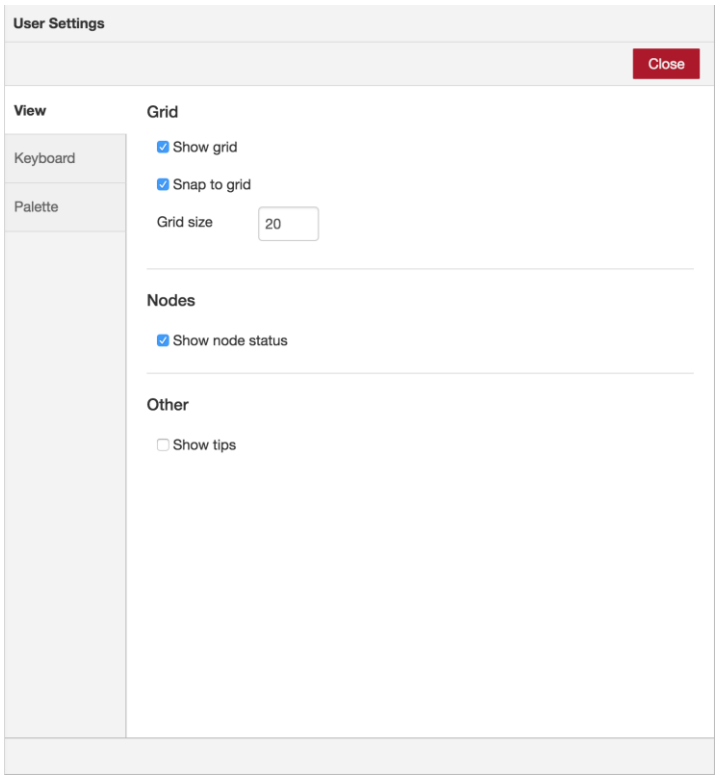


Рис. вкладка «User Settings»

Потоки (Flows)

Рис. Вкладки потоку

Додавання потоку

Щоб додати новий потік натисніть + що знаходиться на верхній панелі

Reference	
Key shortcut	*none*
Menu option	Flows -> Add
Action	core:add-flow

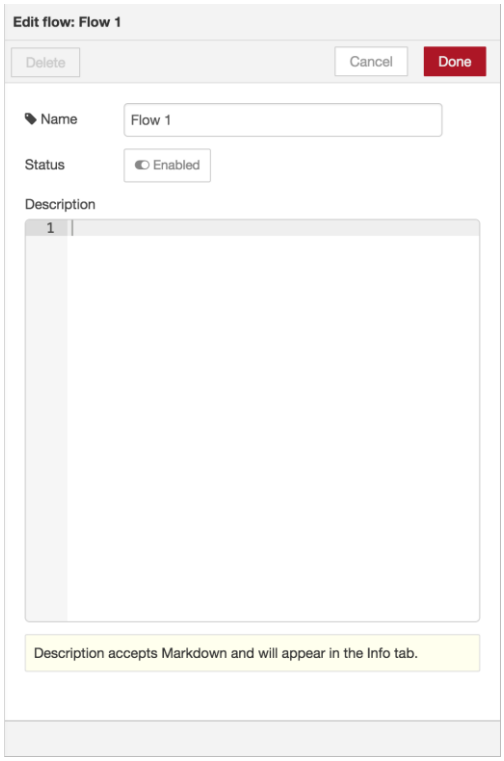


Рис. Редагування властивостей потоку

Щоб змінити властивості потоку двічі натисніть на його вкладці у верхній панелі. Відкриється діалогове вікно Flow Properties.

Reference	
Key shortcut	*none*
Menu option	`Flows -> Rename`
Action	`core:edit-flow`

У діалоговому вікні може бути встановлена назва (name) та опис (description) потоку. Для форматування опису може використовувати синтаксис Markdown. Опис з'явиться у бічній панелі.

Для відключення або включення потоків можна використовувати властивість Status.

Видалення потоків

Щоб видалити потік, натисніть кнопку «Delete» у діалоговому вікні Flow Properties.

Reference	
Key shortcut	*none*
Menu option	`Flows -> Delete`
Action	`core:remove-flow`

Вузли (Nodes)

Вузли можуть бути додані до робочої області такими шляхами:

- Перетягуванням їх з палітри,
- Використовуючи діалогове вікно швидкого додавання,
- Імпортуючи з бібліотеки чи буферу обміну.

рис. Мітки порту вузла

Вузли приєднуються один до одного за допомогою з'єднань через їхні порти. Вузол може мати не більше одного вхідного порту, але багато вихідних. Порт може мати мітку (label), що буде показуватися при наведенні курсору. У вузлі можуть бути вказані спеціальні мітки, наприклад, вузол Switch(перемикач) показує правило що відповідає даному порту. Мітки можна налаштувати в діалоговому вікні редагування вузла.

Деякі вузли відображають статусне повідомлення або піктограми біля вузла. Це використовується для позначення стану вузла в режимі виконання. Наприклад, вузли MQTT вказують на те, чи підключені вони в даний час.

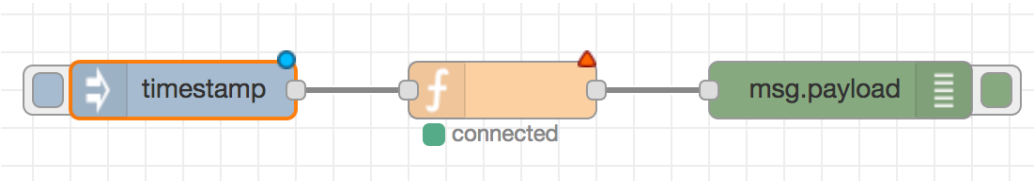


Рис. Елементи вузлів

Якщо вузол має які-небудь зміни, що не були розгорнуті в режимі виконання, це буде відображено синім кружком над ним. Якщо є помилки у конфігурації, то буде відображатися червоний трикутник.

Деякі вузли містять кнопку з лівого або правого боку. Вони служать для взаємодії з вузлом в редакторі. Вузли Inject і Debug є єдиними основними вузлами, які мають такі кнопки

Діалог швидкого додавання

Діалогове вікно швидкого додавання - це простий спосіб додати вузол до робочої області, без необхідності перетягувати її з палітри.

Діалогове вікно відкривається шляхом утримування `Ctrl` або `Command` разом з натисканням (лівий клік миші) в робочі області.

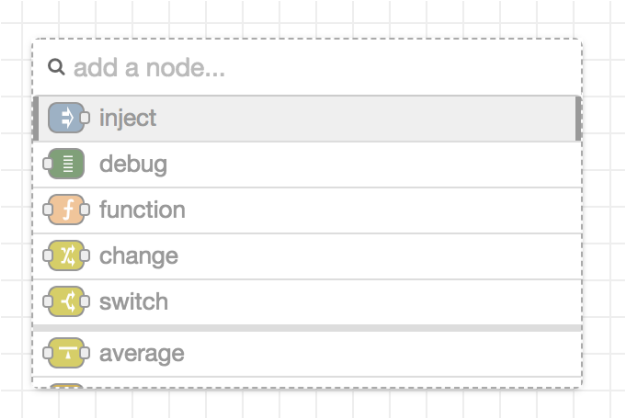


Рис. Діалогове вікно швидкого додавання

У діалоговому вікні міститься повний список всіх вузлів, доступних для додавання. Тут показуються п'ять основних вузлів сервера у верхній частині списку, а потім всі нещодавно додані вузли та, нарешті, повний алфавітний список решти вузлів.

Як і в основній палітрі, у діалоговому вікні можна швидко знаходити вузол або відфільтрувати список.

Редагування конфігурації вузла

Конфігурація вузла може бути відредагована подвійним кліком на вузол чи натисканням клавіші `Enter` коли вузол в робочій області має фокус. Якщо вибрано декілька вузлів, редагуватися почне *перший* з них.

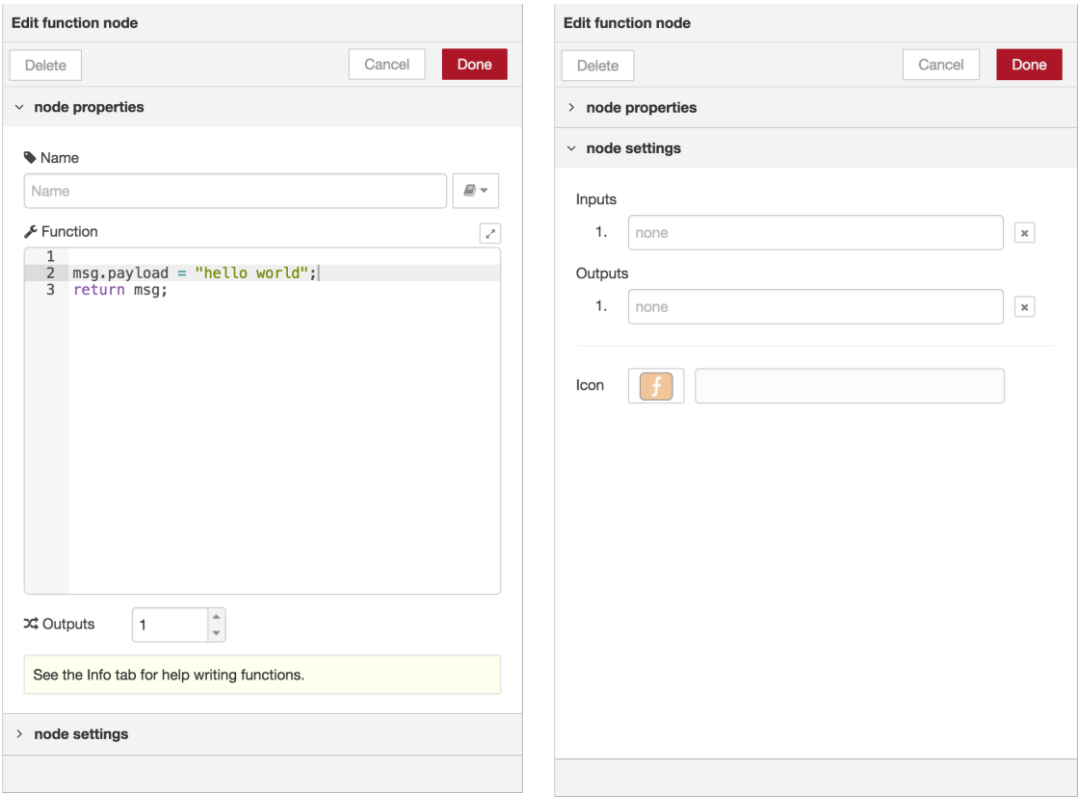


рис. Діалог редагування вузла - розділи властивостей та параметрів

Діалогове вікно редагування вузла має два окремих розділи; властивості (properties) та налаштування (settings). У розділі властивостей відображається форма редагування, специфічна для конкретного типу вузла. У розділі налаштувань відображаються загальні параметри, які

можна встановити на всіх вузлах. Включаючи мітки спеціального порту, а також іконку для вузла.

При натисканні на піктограму бачимо засіб вибору значків вузла, який можна використовувати для вибору іконки вузла зі списку всіх доступних іконок.

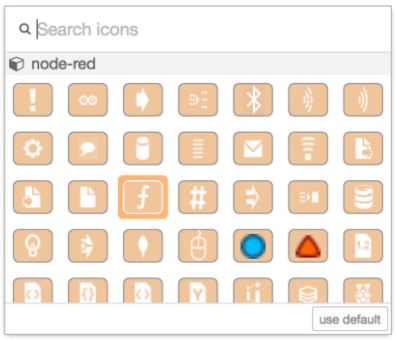


рис. Набір іконок для вузла

Конфігураційні вузли

Конфігураційні вузли (config Node) - це спеціальний тип вузла, що містить конфігурацію багаторазової доступності, що може бути розподілена між звичайними вузлами потоку. Наприклад, вузли «MQTT In» і «MQTT Out» використовують конфігураційний вузол «MQTT Broker» для представлення спільного підключення до брокера MQTT.
Конфігураційні вузли додаються через діалогове вікно редагування вузла, що вимагає вузол конфігурації. У ньому буде поле для вибору з наявних вузлів конфігурації потрібного типу або додати новий екземпляр.

рис. Додавання вузла конфігурації

Натиснувши кнопку поруч із полем вибору, відкриється діалогове вікно редагування для вибраного вузла або додано новий екземпляр.

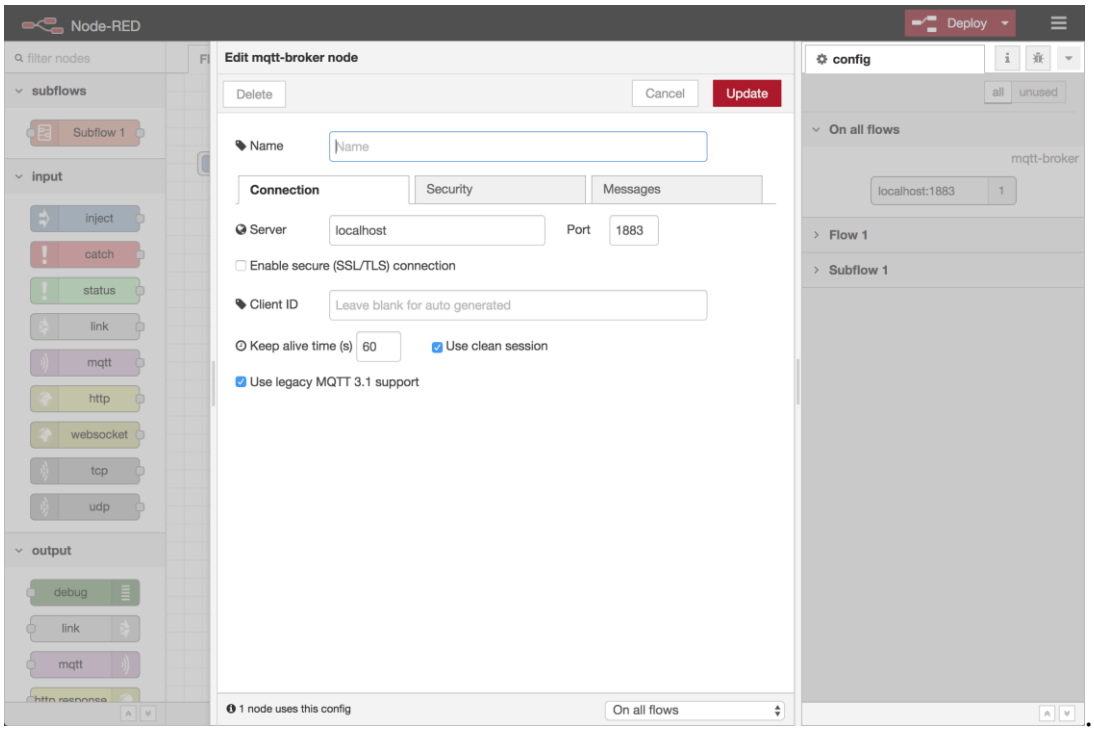


рис. Вікно редагування вузла конфігурації

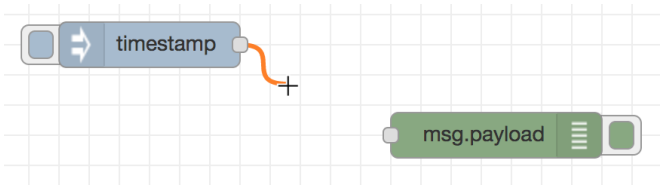
У діалоговому вікні редагування конфігураційного вузла є лише розділ властивостей вузла, оскільки вузол конфігурації не має піктограм або портів для встановлення міток.

У нижній частині діалогового вікна вказується, скільки вузлів використовує цей конфігураційний вузол. Вона також містить поле вибору для означення області застосування конфігураційного вузла. Обсяг визначає те, який поточний вузол конфігурації доступний. За замовчуванням він доступний для всіх потоків, але вікно вибору може бути використане для того, щоб зробити його локальним лише для одного потоку.

Бічна панель конфігурації вузлів може використовуватись для керування всіма конфігураційними вузлами.

З'єднання (Wires)

Вузли з'єднують разом, шляхом натискання на ліву кнопку миші по порту вузла, перетягуючи так званий дріт до кінцевого вузла та відпускаючи кнопку миші.



Альтернативно, якщо клавіша Ctrl/Command утримується, натиснути (і відпустити) лівою кнопкою миші на порт вузла, а потім натиснув на пункт призначення.

Якщо Ctrl/Command залишається утримуваним, а безпосередньо провідний вузол призначення має вихідний порт, тоді з цього порту запускається новий дріт. Це дозволяє швидко з'єднуватися багатьом вузлам разом.

Це також можна поєднати з діалогом швидкого додавання, яке спрацьовує Ctrl / Command-Click на робочому місці, для швидкої вставки нових вузлів і вони вже будуть підключені до попередніх вузлів потоку.

Розділення з'єднання

Якщо вузол із вхідним та вихідним портом перетягнений до середньої точки дроту, дріт прийме вигляд штрихової лінії. При відпусканні він автоматично вставляється в потік в цій точці.

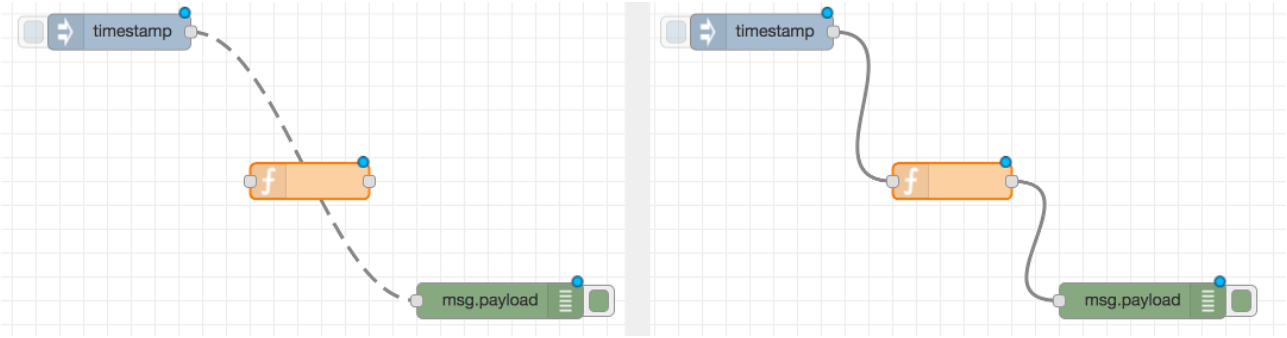


рис. Перемістіть вузол на з'єднання(дріт), щоб вставити його до потоку.

Переміщення з'єднань

Щоб від'єднати дріт від порту, виберіть дріт, натисніть на нього і потримайте клавішу Shift. Після цього дріт відключається від порту і може бути скинутий на інший порт. Якщо відпустити кнопку над робочою панеллю, дріт буде видалено.

Якщо до порту є декілька дротів, підключених до нього і жоден з них не буде виділений, коли натискається Shiftt всі дроти почнуть рухатися.

Видалення з'єднань

Щоб видалити дріт, спочатку виберіть його, натиснувши на нього, а потім натисніть кнопку delete

Під-потоки (Subflows)

Під-потоки - це сукупність вузлів, які згортаються в єдиний вузол у робочій області.

Вони можуть бути використані для зменшення певної візуальної складності потоку або для об'єднання групи вузлів, що використовується в різних місцях.

Після створення, під-потік додається до палітри доступних вузлів. Потім окремі екземпляри під-потоку можна додати до робочої області, як і будь-який інший вузол.

Примітка: під-потік не може існувати сам собою без жодних вузлів, він повинен містити їх прямо або опосередковано.

Створення порожнього під-потоку

Під-потік можна створити, вибравши в меню пункт 'Subflow -> Create subflow' Це створить порожній під-потік і відкриває його у робочій області.

Reference	
Key shortcut	*none*
Menu option	Subflows -> Create subflow
Action	core:create-subflow

Перетворення вузлів в під-потоки

Також можна перетворити поточний вибір вузлів на підрівень, вибравши в меню пункт ‘Subflow -> Selection to Subflow’. Ці вузли будуть переміщені в новий під-потік і замінені на його екземпляр у потоці.

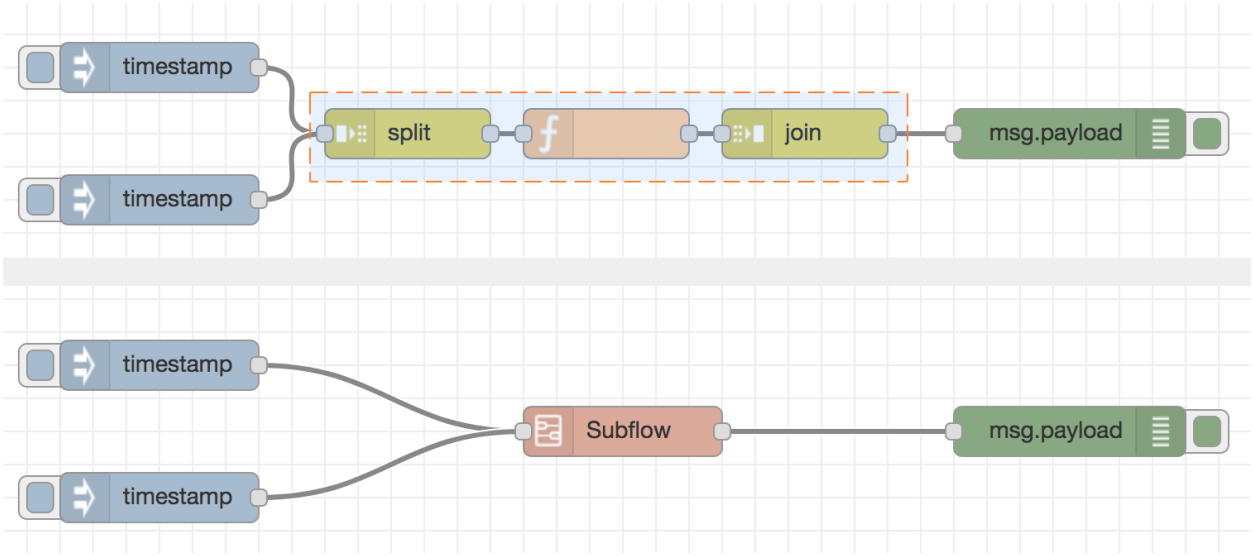


рис. Створення під-потоку

Це можливо тільки в тому випадку, якщо вхідні з’єднання підключені до одного вузла, - тому що результуючий під-потік вузла сам може мати не більше одного входу.

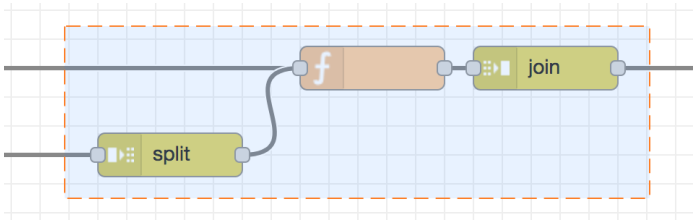


рис. Недійсний вибір для під-потоку

Reference	
Key shortcut	*none*
Menu option	Subflows -> Selection to Subflow
Action	core:convert-to-subflow

Редагування під-потоку

Існує два способи відкрити під-потік для редагування його вмісту. Двічі клацнути на вузол у палітрі або натискання кнопки ‘Edit flow template’ у діалоговому вікні редагування вузла

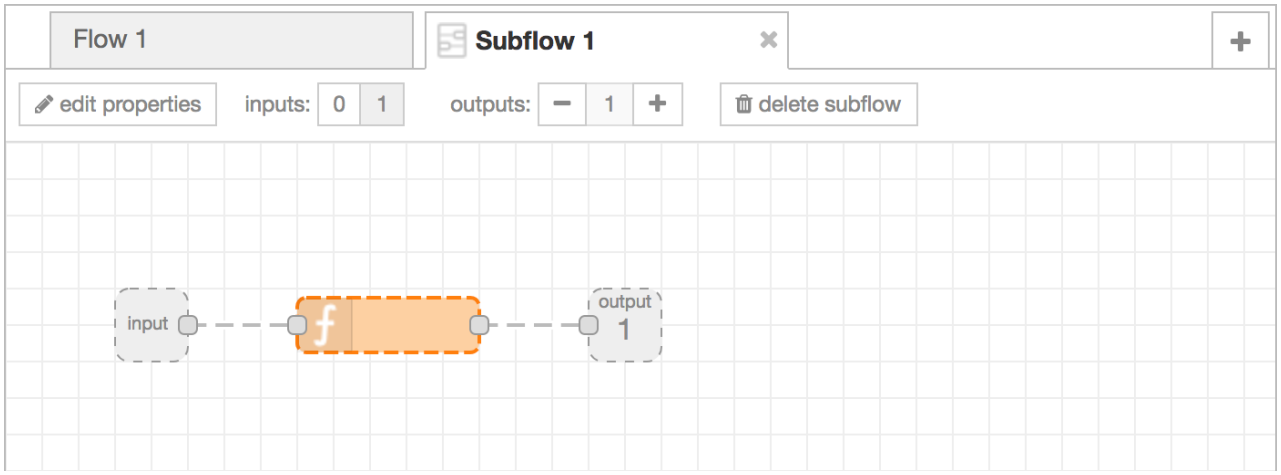


Рис. Редагування під-потоку

Під-потік відкривається в робочій області як нова вкладка. На відміну від звичайних вкладок потоку, вкладки під-потоків можуть бути закриті, щоб приховати їх.

Входи & Виходи

Входи та виходи під-потоку представлені сірими квадратними вузлами, які можуть бути підключені до потоку. Панель інструментів надає можливість додавання та видалення цих вузлів. Як і в звичайних потоках у вузлів, може бути не більше одного вводу та багато виходів, за необхідності.

рис. Вікно редагування властивостей під-потоку

Властивості під-потоку

Кнопка ‘edit properties’ відкриває вікно властивостей під-потоку. Як і у діалоговому вікні властивостей потоку, тут можна вказати назву та опис під-процесу.

Можна також встановити категорію під-потоку, або шляхом вибору з однієї з існуючих категорій або додати нову.

Видалення під-потоку

Кнопка ‘delete subflow’ на панелі інструментів під-потоку може бути використана для видалення під-потоку та всіх його екземплярів.

Вибір вузла (Selection)

Вузол вибирається при натисканні наньому. Це призведе до скасування вибору будь-якого іншого елемента. Бічна панель інформації буде оновлюватися, щоб відобразити властивості вузла та текст довідки для його типу.
Якщо Ctrl або Command тримається при натисканні на вузол, вузол буде доданий до поточного вибору (або видалений, якщо він вже був обраний).
Якщо при натисканні на вузол утримується Shift , він вибере цей вузол та всі інші вузли, до якого той підключений.
Дріт (з’єднання) вибирається при натисканні на ньому. На відміну від вузлів, можна вибрати лише одне з’єднання за один раз.

Інструмент «ласо»

Інструмент «ласо» може використовуватися для вибору кількох вузлів. Він активується за допомогою перетягування одночасно з натисканням в робочій області. Його не можна використовувати для вибору з’єднання.

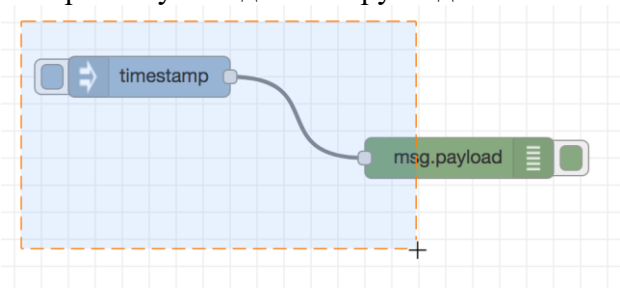


рис. Вибір декількох вузлів за допомогою lasso

Вибір всіх вузлів

Щоб вибрати всі вузли потоку переконайтеся що робоча область має фокус, а потім натисніть Ctrl/Command-a.

Reference	
Action	core:select-all-nodes
Key shortcut	Ctrl/⌘-a

Редактор буфер обміну

Редактор підтримує такі дії, як копіювання/вирізання/вставка. Зверніть увагу, що дії використовують внутрішній буфер обміну, а не системний.

Reference	
Action	`core:copy-selection-to-internal-clipboard`
Key shortcut	`Ctrl/⌘-c`

Reference	
Action	`core:cut-selection-to-internal-clipboard`
Key shortcut	`Ctrl/⌘-x`

Reference	
Action	`core:paste-selection-from-internal-clipboard`
Key shortcut	`Ctrl/⌘-v`

Імпорт та експорт потоків (Importing and Exporting Flows)

Потоки можна імпортувати та експортувати з редактора, використовуючи формат JSON, що дозволяє дуже легко обмінюватися потоками з іншими.

Імпорт потоків

Щоб імпортувати потік, відкрийте діалогове вікно «Import»-> “Clipboard”, вставте json та натисніть «Import to Current flow».

Кнопка ‘Import’ активна лише тоді, коли у діалогове вікно вставлений правильний JSON.

У діалоговому вікні також є можливість імпортувати вузли в поточний потік або створити для них новий потік

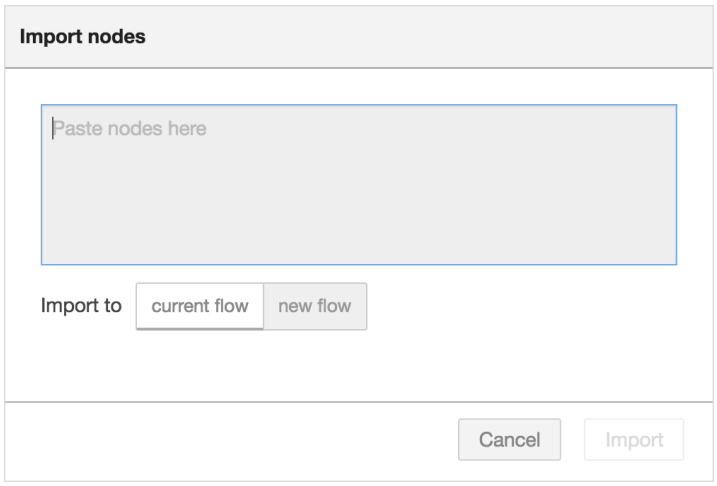
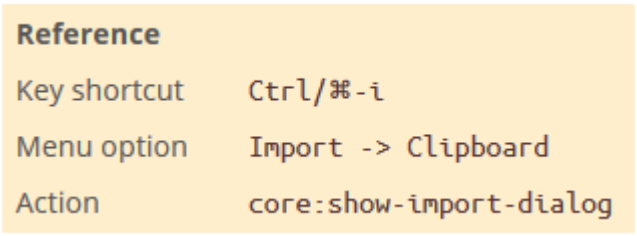


рис. Діалог імпорту потоків



Експорт потоків

Вікно експорту може бути використано для копіювання потоку json з редактора.

Можна експортувати виділені вузли, поточний потік(включаючи його вузли) або повну конфігурацію потоку.

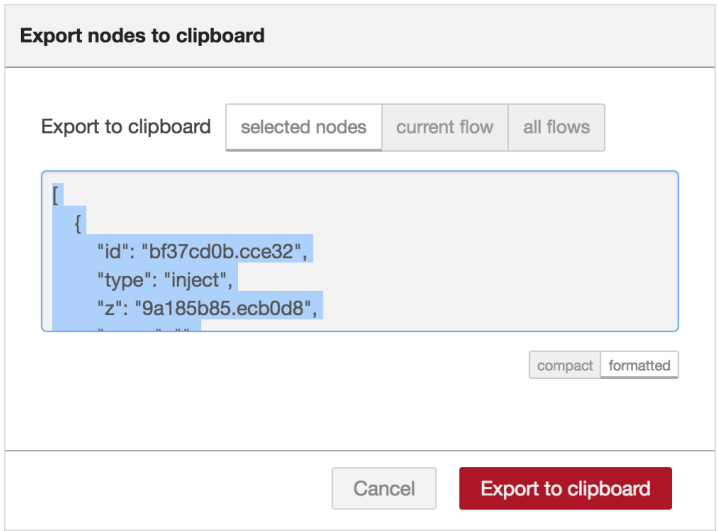


рис. Діалог Export Flows

У діалоговому вікні експорту є можливість вибрати експорт в компактний або відформатований JSON. Опція «compact» означує створення єдиного рядку JSON без пробілів. Опція «formatted» робить JSON форматованим по декільком рядкам з повним відступом, що є зручним для читання.

Пошук потоків (Searching flows)

Діалогове вікно пошуку може використовуватися для пошуку вузлів усередині робочої області, включаючи конфігураційні вузли.

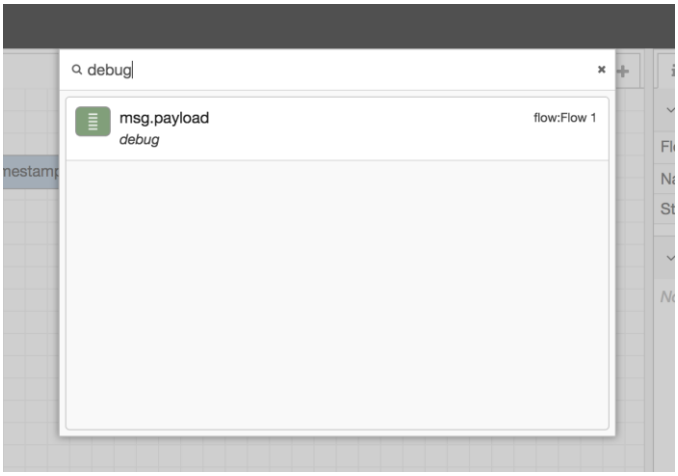


рис. Діалог пошуку

Він індексує всі властивості вузлів, тому його можна використовувати для пошуку вузла за його ідентифікатором, типом, ім'ям або будь-якою іншою властивістю.

Reference	
Key shortcut	Ctrl/⌘-f
Menu option	Search flows
Action	core:search

Вибір вузла у списку результатів покаже той вузол у редакторі.

Палітра (Palette)

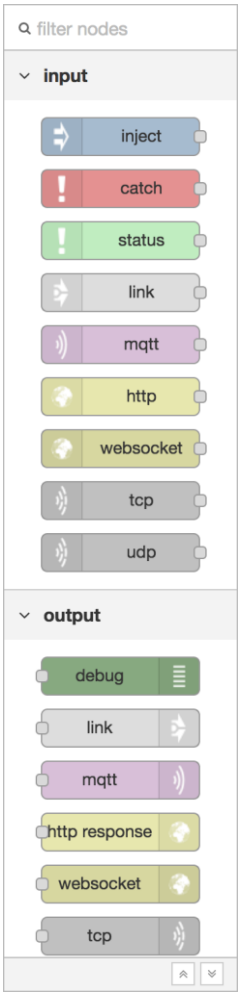
Палітра містить всі вузли, які встановлені та доступні для використання.

Вони організовані в декілька категорій починаючи зверху з inputs, outputs та functions. Якщо є під-потоки, вони з'являються у категорії у верхній частині палітри.

Категорії можна розширити або згорнути, натиснувши заголовок.

Кнопки ⌃ та ⌄ в нижній частині палітри можуть бути використані для згортання або розширення всіх категорій.

Над палітрою є поле для введення, яке можна використовувати для фільтрації списку вузлів.



Менеджер палітри (Palette Manager)

Менеджер палітри може використовуватися для встановлення нових вузлів до палітри. Доступ до нього можна отримати за вкладкою Palette tab в User Settings dialog.

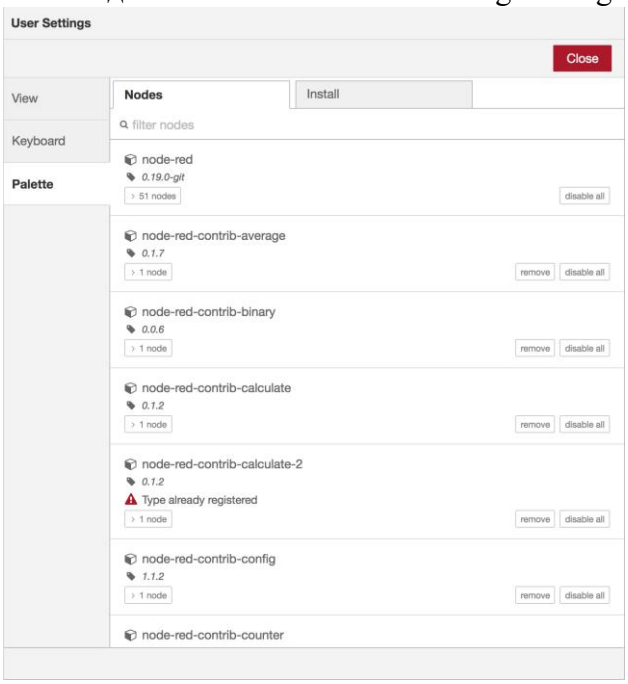


рис. Palette Manager - Nodes tab

Менеджер палітри має дві вкладки:

- Nodes lists - список вузлів, які вже встановлені у середовищі виконання
- Install lists - список вузлів доступних до встановлювання

Reference	
Key shortcut	Ctrl/⌘-Shift-p
Menu option	Manage palette
Action	core:manage-palette

Керування вузлами

Кожен запис у списку Nodes list відображає назву та версію модуля, а також список окремих типів вузлів, які надає модуль.

Параметри надаються для видалення, відключення або оновлення кожного модуля.

Якщо вузол в даний час використовується в потоці, модуль неможливо видалити чи деактивувати в палітрі

Встановлення вузлів

Вкладка Install може використовуватись для пошуку доступних модулів та їх встановлення.

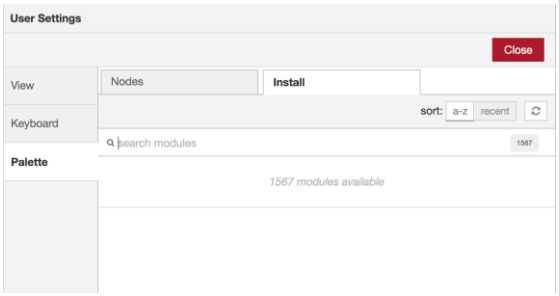


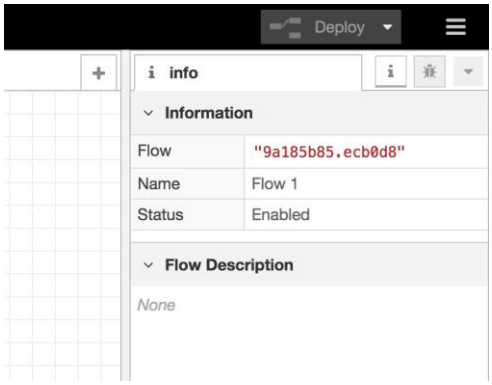
рис. Palette Manager - Install tab

Щоб знайти модуль, введіть його ім'я на панелі пошуку. Результати пошуку показують деталі модулів, в тому числі, коли востаннє було оновлено, і посилання на його документацію. Його можна встановити, натиснувши кнопку ‘install’.

рис. Palette Manager - Install module details

Бічна панель (Sidebar)

Деякі вузли додають власні панелі бічної панелі, наприклад node-red-dashboard.



Editor Sidebar

Панель відкриється якщо натиснути на значок в заголовку бічної панелі або вибравши їх з випадаючого списку.
Можна змінити розмір бічної панелі, перетягнувши її край робочої області.
Якщо край перетягнути близько до правого краю, бічна панель буде прихована. Це може бути показано знову, вибравши опцію ‘Show sidebar’ в меню View menu, чи через комбінацію Ctrl/⌘-Space.

Reference	
Key shortcut	Ctrl/⌘-Space
Menu option	View -> Show sidebar
Action	core:toggle-sidebar

Бічна панель: Інформація (Information)

На бічній панелі Information відображається більше інформації про поточний вибраний вузол, зокрема:

- Перелік його властивостей
- Текст довідки до вузла

Якщо нічого не вибрано, він відображає опис поточного потоку, який можна редагувати в Flow Properties edit dialog.

Reference	
Action	core:show-info-tab
Key shortcut	Ctrl/⌘-g i

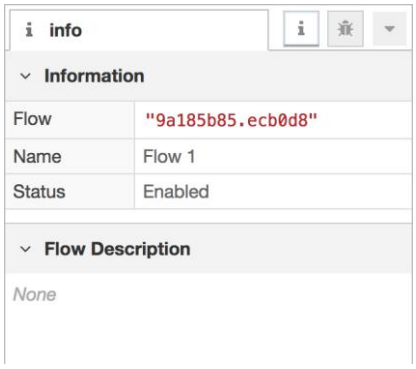
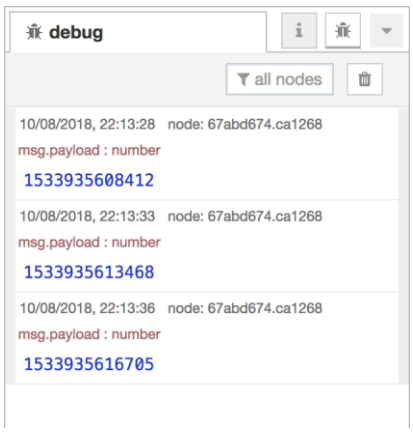


рис. Information Sidebar

Бічна панель: Debug messages

На бічній панелі Debug відображаються повідомлення, передані вузлам Debug в потоці, а також певні повідомлення журналу середовища виконання (runtime).



Для отримання додаткової інформації про те, як використовувати бічну панель Debug та для розуміння структури повідомлень, прочитайте посібник Working with messages .

Reference	
Action	core:show-debug-tab
Key shortcut	Ctrl/⌘-g d

За замовчуванням бічна панель Debug відображає всі передані йому повідомлення. Повідомлення можна фільтрувати в панелі параметрів фільтрів, що відкривається відповідною кнопкою.

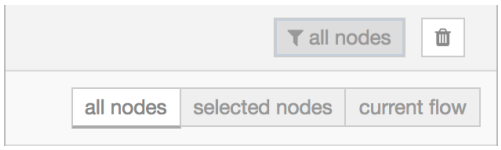


рис. Параметри фільтра налагоджування

Панель містить 3 опції:

- *all nodes* - відображає всі повідомлення
- *selected nodes* - виберіть певні вузли відлагодження зі списку всіх доступних вузлів
- *current flow* - відображає лише повідомлення з вузлів поточного потоку у робочій області

Примітка: бічна панель Debug може відображати лише 100 останніх повідомлень. Якщо бічна панель наразі показує фільтрований список повідомлень, приховані повідомлення входять до 100 обмежених. Бічна панель може бути очищена через певний час натисненням на кнопку. Якщо потік має вузли Debug, які створюють багато повідомлень (флуд), і не фільтруються з бічної панелі, то краще їх відключити, натиснувши їхню кнопку в робочій області.

Кнопка в нижньому колонтитулі бічної панелі можна використовувати для відкриття окремого вікна веб-переглядача, що містить бічну панель Debug.

Бічна панель: Конфігураційні вузли (Configuration nodes)

Бічна панель конфігураційних вузлів (Configuration nodes) надає список всіх конфігураційних вузлів в одному екрані. Кожен вузол показує його тип і мітку, а також підрахунок того, скільки вузлів поточного потоку використовує цей вузол конфігурації.

Якщо конфігураційний вузол не використовується, він відображається пунктиром. Переглядач також можна фільтрувати, щоб показати лише невикористані вузли, вибравши у заголовку фільтр "unused ". У діалоговому вікні редагування подвійним кліком миші можна відкрити конфігураційний вузол.

Reference	
Action	core:show-config-tab
Key shortcut	Ctrl/⌘-g c

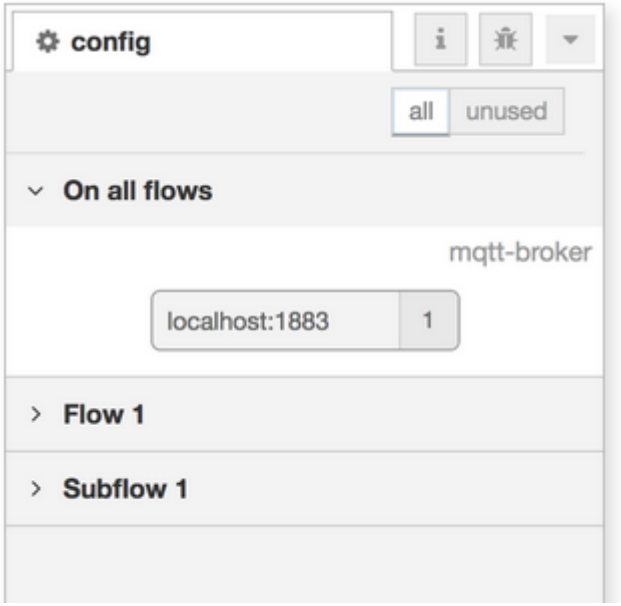


рис. Конфігураційні вузли бічної панелі

Бічна панель: Контекстні дані (Context data)

Бічна панель контексту відображає вміст сховища контекстних даних. Щоб отримати додаткові відомості про використання контексту, прочитайте Робота з контекстом посібник.

Reference	
Action	core:show-context-tab
Key shortcut	Ctrl/⌘-g x

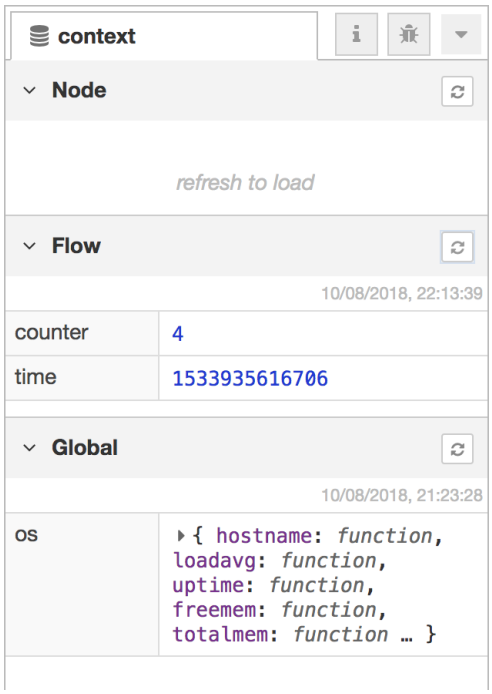


рис. бічна панель контекстних даних

Панель розділена на три частини, по одному для кожного типу контексту.

Розділ 'Node' показує контекст поточного виділеного вузла. Він не відображає вміст автоматично, а вимагає, щоб користувач натиснув кнопку оновлення, щоб завантажити його. Розділ 'Flow' показує контекст поточного потоку. Він автоматично оновлюється кожного разу, коли потік змінюється в основній робочій області. У розділі «Global» відображається глобальний контекст і завантажується кожного разу, коли завантажується редактор.

Для всіх трьох областей, щоб побачити зміни, потрібно натиснути відповідну кнопку оновлення. Наведіть курсор миші на будь-яке ім'я контекстної властивості для відображення кнопки оновлення, яка може бути використана для оновлення лише одного значення.

Наведіть курсор на значення контекстного ресурсу для відображення кнопки, яка може скопіювати його вміст у системний буфер обміну. Значення буде перетворено в JSON, тому не всі значення можуть бути скопійовані.

Проекти (Projects)

Представлення про проекти (Introducing projects)

Проекти є новим способом керування файлами потоку. Замість того, щоб трактувати свої потоки як просту пару файлів, вони представляють все необхідне для створення програми Node-RED що може бути розповсюджена.

Проекти підтримуються репозиторієм Git, що означає, що усі файли повністю керовані версією та дозволяють розробникам використовувати знайомі робочі процеси для співпраці з іншими.

У версії 0.18 функція проектів знаходиться в режимі попереднього перегляду. Це означає, що він повинен бути включений у файлі налаштувань.

Ця функція наразі недоступна в середовищі IBM Cloud.

Активація функції проектів (Enabling projects)

Щоб увімкнути функцію проектів, відредагуйте свій файл settings.js і додайте наступну опцію в блок module.exports і перезапустіть Node-RED.

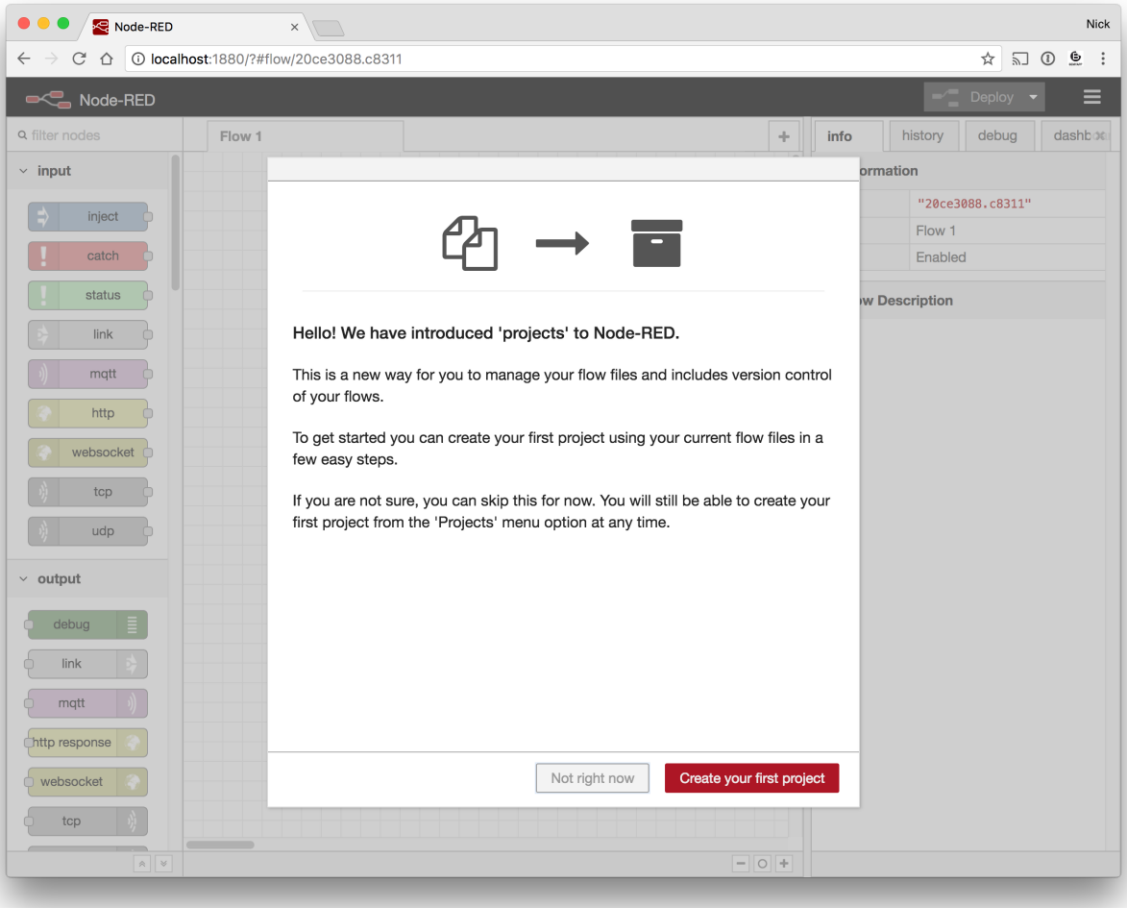
```
   editorTheme: {
       projects: {
           enabled: true
       }
   },
```

Примітка : файл settings.js експортує *об'єкти JavaScript*. Для конфігурування Node-RED, ви повинні зрозуміти, як змінити об'єкт JavaScript, додавши нові або змінюючи існуючі пари ключів / значень, такі як editorTheme що наведені нижче.

Ця функція залежить від наявності утиліт командного рядку git та ssh-keygen. Node-RED перевірить їх на запуск і повідомить вас, якщо вони відсутні.

Створення вашого першого проекту (Creating your first project)

Коли ви відкриваєте редактор, то бачите екран привітання, який запрошує вас створити свій перший проект за допомогою ваших файлів потоку.



Це допоможе виконати наступні кроки:

1. **Налаштуйте клієнт керування версіями**

Node-RED використовує інструмент з відкритим кодом Git для керування версіями. Він відстежує зміни ваших файлів проекту і дозволяє користуватися ними на дистанційних сховищах - репозиторіях (типу RDP)

Коли ви здійснюєте набір змін, Git записує хто вніс зміни з іменем користувача та електронною адресою. Ім'я користувача може бути будь-яке.

Ви вже можете налаштувати свій Git-клієнт - в якому Node-RED буде повторно використовувати ці налаштування.

Ви можете змінити ці налаштування в будь-який час за допомогою головного діалогу настройок Node-RED.

- Докладніше про налаштування Git-клієнта (GitHub)

2. **Створіть свій проект**

Наступний крок дає змогу назвати ваш проект та надати йому опис.

3. **Створіть файли проекту**

Node-RED автоматично перенесе ваші файли потоків у ваш проект. Ви можете перейменувати їх тут, якщо хочете.

4. **Налаштування шифрування файлу облікових даних**

Так як ви можете поділитися своїм проектом на публічних сайтах, таких як GitHub, в такому випадку дуже рекомендується шифрувати файл облікових даних

Щоб його зашифрувати, потрібно вибрати ключ, який буде використовуватися для захисту файлу. Цей ключ не зберігається в межах проекту. Якщо хтось клонує ваш проект, вам потрібно буде надати їм ключ для розшифровки файлу облікових даних. Інакше їм доведеться відредагувати потоки, щоб надавати власні облікові дані.

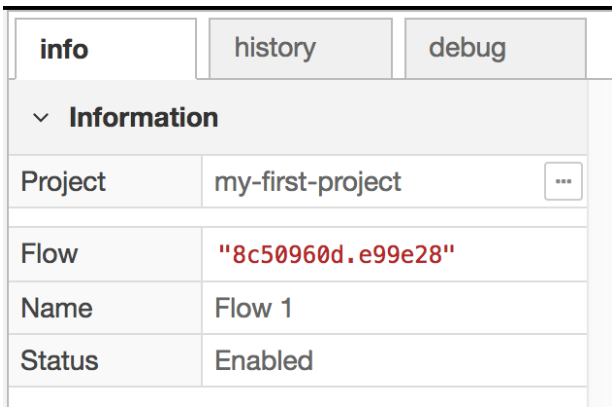
Потім проект створюється у каталозі: ~/.node-red/projects/<project-name>.

Робота з проектами (Working with projects)

Створивши свій проект, ви можете продовжувати користуватися редактором Node-RED, як і завжди. Є кілька нових частин редактора, які були додані для роботи з вашим проектом.

Доступ до налаштувань проекту

Бічна панель інформації у верхній частині сторінки показує, з яким проектом ви працюєте. Біля назви проекту є кнопка, яка відкриває діалогове вікно Параметри проекту - Project Settings



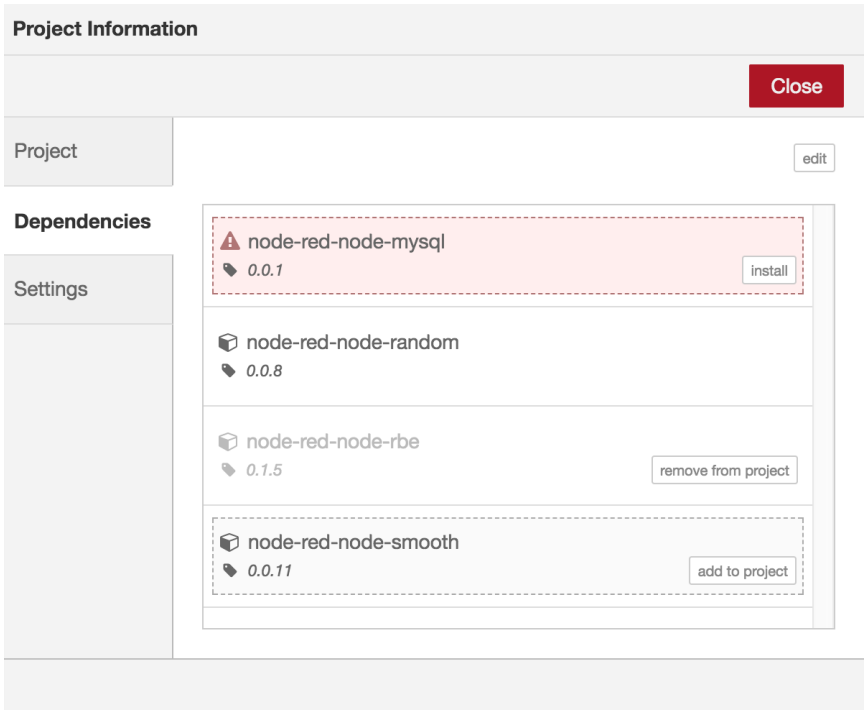
Ви також можете отримати доступ до нього через опцію в головному меню Projects -> Project Settings

Діалогове вікно має три вкладки:

- Project - дозволяє редагувати файл README.md проекту..
- Dependencies - керувати списком модулів вузлів, на які залежить ваш проект
- Settings - керувати налаштуваннями проекту, включаючи GIT керування

Project Dependencies

Кожен проект має свій власний файл package.json, який містить список модулів вузлів, від яких залежить проект. Редактор Node-RED відстежує, які вузли ви використовуєте в потоці, і допомагає вам підтримувати цей список залежностей.



На знімку екрана вище, поточний проект має три модулі, перераховані в його файлі `package.`, кожен в різному стані:

- node-red-node-mysql наразі не встановлено
- node-red-node-random використовується поточним потоком
- node-red-node-rbe перераховано, але не використовується поточним потоком

І нарешті, node-red-node-smooth забезпечує вузол, який використовується поточним потоком, але цей модуль не вказаний як залежний. Необхідно оновлювати список залежностей, якщо ви хочете поділитися проектом з іншими - так як це допоможе користувачам встановити необхідні модулі.

Параметри проекту

Вкладка параметрів проекту (project settings) дозволяє вам керувати файлами потоку, конфігурацією шифрування ваших облікових даних та налаштовувати локальні Git гілки та віддалені репозитарії.

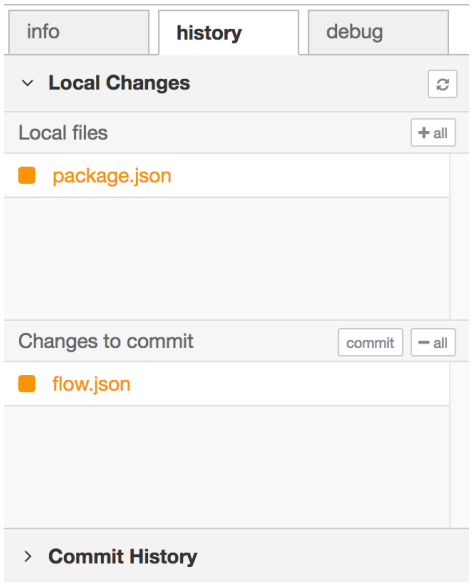
Контроль версії

До бічної панелі була додана нова вкладка history. Тут ви керуєте контролем над версією вашого проекту. Вкладка має два розділи:

- Local Changes - показує файли проекту, які змінилися, дозволяючи виставити і впорядковувати їх.
- Commit History - список всіх коммітів у репозитарії, з інструментами для push коммітів до віддалених репозитаріїв.

Локальні зміни

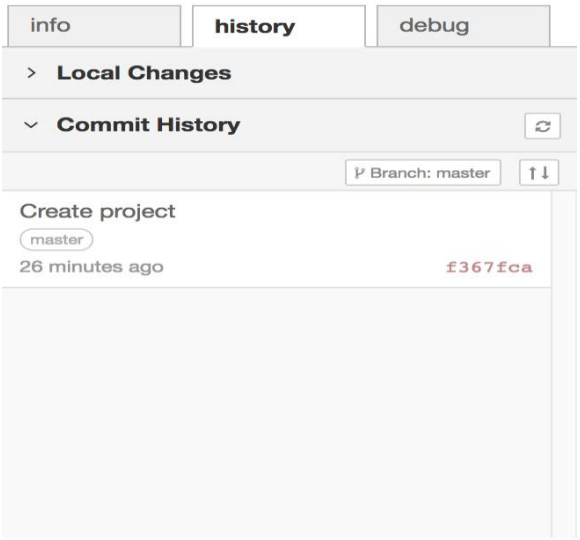
Кожного разу, коли ви змінюєте файл проекту, наприклад, розгорнувши нову конфігурацію потоку, вона буде перелічена в розділі ‘Local files’ Ви можете натиснути на назву файлу, щоб побачити, як відрізняється зміна. Коли ви наведете курсор миші на файл, ви побачите кнопку «+» - натиснувши на яку він переміститься до списку ‘Changes to commit’.



Коли ви встановите файли, з якими хочете працювати, натисніть кнопку commit введіть повідомлення та підтвердьте.

Commit History

У розділі Commit History перелічено всі комміти у поточній гілці сховища. Під час створення проекту Node-RED автоматично створює початковий набір файлів за замовчуванням для проекту.



У верхній частині списку знаходиться кнопка ‘Branch’ що дозволяє вам перевірити / створити гілки в сховищі.

Якщо у вашому репозитарії є налаштований дистанційний репозитарій, то є також кнопка, яка показує, скільки новіших та/або новіших коммітів у вашому локальному репозитарію, порівняно з дистанційним. Це дозволяє вибрати дистанційне/відгалужене відстеження і натискання/перетягування змін до пульта дистанційного керування.

If your repository has a remote repository configured, there is also a button that shows how many commits ahead and/or behind your local repository is compared with the remote. It allows you to pick the remote/branch to track, and push/pull your changes to the remote.

Це одна з областей, в якій редактор Node-RED намагається спростити дію користувачу і не надає всього набору опцій, що надає git. Це область, в якій ми раді отримати зворотній зв'язок. Наприклад, він не надає опцій для перебазування місцевих коммітів, або примусово не вносить зміни до дистанційного репозиторію. Та ви все ще можете робити ці речі, повернувшись до командного рядка.

Створення нових проектів

Після того, як ви створили свій перший проект, перенісши існуючі файли потоку, ви можете створити додаткові проекти.

Оберіть з меню `Projects -> New` відкривається діалогове вікно Projects

Є три доступних опції:

- відкрити існуючий проект
- створити новий проект
- клонування репозитарію проекту

Відкрити існуючий проект

Node-RED може виконувати лише один проект в один момент часу. Відкриваючи інший проект, ви змінюєте запущені потоки

Переглядач ‘open project’ також дозволяє видаляти проекти: наведіть курсор на проект у списку та натисніть кнопку видалення. Ви не можете видалити активний проект.

Створити новий проект

Це дозволяє створити новий проект. Тут ті самі опції, що й в наборі екранів ‘create your first project’, але об’єднані в один.

Клонування репозитарію проекту

Це дозволяє клонувати існуючий віддалений репозиторій. Ви можете використовувати або http(s) або git/ssh url для сховища. Якщо репозитарій вимагає автентифікації, ви повинні пройти його тут

Примітка: для http url-адреси, не включають ім’я користувача і його пароль. Ви маєте надати їх окремо, коли з’явиться запит.

Node-RED наразі не використовує ніяких помічників облікових даних, з якими ви можете налаштувати клієнт git. Це область, на яку ми звертаємо увагу спільноти.

Для git/ssh, Node-RED запропонує список доступних SSH відкритих ключів. Цей список генерується скануванням ~/.ssh для файлів з розширенням .pub що мають відповідні файли приватних ключів. Це також дозволяє генерувати нові пари публічних / приватних ключів через вкладку ‘Git config’ вікна основних налаштувань. Ці файли зберігаються в ~/.node-red/projects/.sshkeys/. Якщо в вас увімкнений adminAuth файли визначаються для поточного входу в систему, тому їм не потрібно передавати облікові дані для дистанційного репозитарію.

Основні вузли (The Core Nodes)

Палітра Node-RED включає стандартний набір вузлів, які є основними будівельними блоками для створення потоків. Цей розділ висвітлює основний набір, про який ви повинні знати.

Всі вузли включають документацію, яку ви можете переглянути на вкладці Info бічної панелі коли вибираєте вузол.

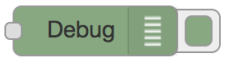
Inject 

Вузол **Inject** може бути використаний для ручного запуску потоку, натиснувши кнопку вузла в редакторі. Він також може використовуватися для автоматичного запуску потоків через регулярні інтервали часу.

Повідомлення, надіслане вузлом Inject може мати його набір властивостей payload (корисне навантаження) і topic (тема).

Для payload можна встановити різні типи:

- значення контексту потоку (flow context) або глобального контексту (global context)
- рядок, число, логічне значення, Buffer або Object
- відмітку часу (Timestamp) – значення в мілісекундах з 1 січня 1970 року

Debug 

Вузол Debug може ви використовуватися для відображення повідомлень на бічній панелі Debug у редакторі.

Бічна панель забезпечує структурований перегляд повідомлень, що надсилаються, що полегшує вивчення цього повідомлення.

Поряд з кожним повідомленням бічна панель налагодження включає в себе інформацію про час надходження повідомлення та місце з якого вузлу воно надіслане. Натискання на ідентифікатор вихідного вузла покаже цей вузол у робочій області.

Кнопку на вузлі можна використовувати для ввімкнення або вимкнення його виходів. Рекомендується відключити або видалити будь-які вузли Debug, які не використовуються.

Цей вузол також може бути налаштований для відправлення всіх повідомлень до журналу середовища виконання, або для надсилання коротких (до 32 символів) повідомлень про статус налагоджуваного вузла.

Сторінка Working with messages надає додаткову інформацію про використання бічної панелі Debug.

Function 

Вузол Function дозволяє запускати код JavaScript для обробки повідомлень, які передаються через нього.

Повний доступ до керівництва для використання вузла функцій тут.

Change 

Вузол Change можна використовувати для зміни властивостей повідомлення та встановлення властивостей контексту без необхідності вдаватися до вузла Function.

Кожен вузол може бути налаштований з декількома операціями, які застосовуються в такому порядку. Доступні операції::

- **Set** - встановити властивість. Значення може бути з різними типами або може бути взяте з існуючого повідомлення або властивості контексту.
- **Change** - пошук і заміна частин властивості повідомлення.
- **Move** - перемістити або перейменувати властивість.
- **Delete** – видалити властивість.

При встановленні властивості це значення також може бути результатом виразу JSONata . JSONata - це декларативна мова запитів та перетворень для даних JSON.

Switch 

Вузол Switch дозволяє передавати повідомлення до різних гілок потоку, оцінюючи набір правил для кожного повідомлення.

Цей вузол налаштовано за допомогою властивості для тесту - це може бути або властивість повідомлення (message property), або властивість контексту (context property).

Існує чотири типи правил:

- **Value** правила оцінюються для зконфігурованої властивості
- **Sequence** правила можуть бути використані для послідовності повідомлень, наприклад, ті, які створені вузлом Split
- може бути надана JSONata **Expression**, який буде використаний для перевірки всього повідомлення і буде відповідати, якщо вираз повертає значення `true` .
- **Otherwise** правило може бути використане для відповідності, якщо жодне з попередніх правил не співпало.

Вузол переведе повідомлення на всі виходи, що відповідають правилам відповідності. Але він також може бути налаштований таким чином, щоб зупинити перевірку правил одразу, як знайдеться правило, що виконується.

Template 

Вузол Template може використовуватися для створення тексту за допомогою властивостей повідомлення для заповнення шаблону.

Для формування результату він використовує шаблонну мову Mustache .

Наприклад, шаблон:

```
This is the payload: {{payload}} !
```

Замініть {{payload}} з значенням властивості повідомлення payload.

За замовчуванням, Mustache замінить певні символи своїми HTML-кодами. Щоб зупинити це, ви можете скористатися потрійними фіксаторами: {{{payload}}}.

Mustache підтримують прості цикли у списках. Наприклад, якщо msg.payload містить масив імен, таких як: ["Nick", "Dave", "Claire"], то шаблон створить список HTML імен:

```
<ul>
{{#payload}}
  <li>{{.}}</li>
{{/payload}}
</ul>
```

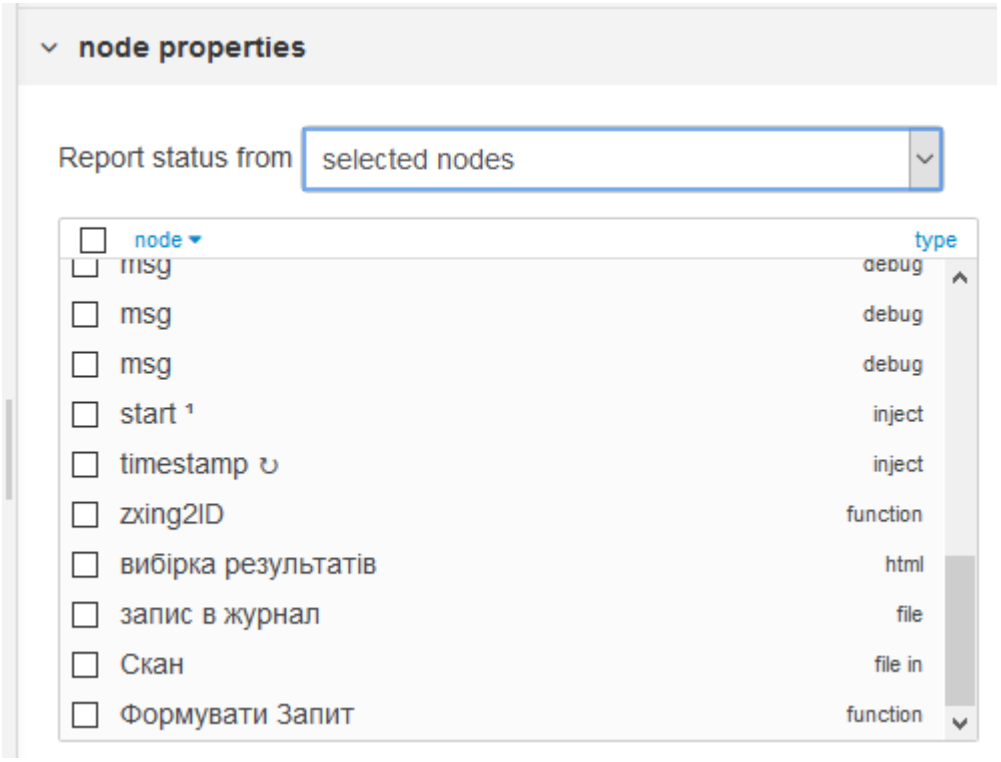
```
<ul>
  <li>Nick</li>
  <li>Dave</li>
  <li>Claire</li>
</ul>
```

Вузол встановить налагоджене повідомлення або контекстну властивість за результатом шаблону. Якщо шаблон генерує дійсний вміст JSON або YAML, його можна налаштувати для аналізу результату на відповідний об'єкт JavaScript.

Вузол Template може використовуватися для створення тексту за допомогою властивостей повідомлення для заповнення шаблону.

Status

Показує про стан (status message) виділених вузлів з потоку. За замовчуванням вузол повідомляє про стан всіх вузлів на одній вкладці робочого простору. Він може бути налаштований для повідомлення про стан окремих вузлів.



Цей вузол не генерує `payload`, він формує об’єкт `status`, з наступними властивостями:
`text` (string) – статусний текст
`source.type` (string) – тип вузла, що передає статус
`source.id` (string) – id вузла, що передає статус
`source.name` (string) – ім’я, якщо є, вузла що передає статус

Написання функцій (Writing Functions)

Вузол Function дозволяє запускати код JavaScript для обробки повідомлень, що передаються через нього.

Повідомлення передається у вигляді об'єкта з назвою `msg`. За замовченням він буде мати властивість `msg.payload` що містить тіло повідомлення.

Інші вузли можуть додавати до повідомлення власні властивості, які повинні бути описані у документації.

Написання функцій (Writing a Function)

Код, введений у вузол Function – це *тіло* функції. Найпростіша функція просто повертає повідомлення:

```
return msg;
```

Якщо тіло повертається null, тоді жодне повідомлення від вузла не передається далі, тобто потік на ньому закінчується.

Повернений функцією об'єкт-повідомлення не повинен бути таким самим типом об'єкту, який був переданий їй; функція може побудувати зовсім новий об'єкт, перш ніж повертати його. Наприклад:

```
var newMsg = { payload: msg.payload.length };
return newMsg;
```

Примітка : будуючи новий об'єкт, повідомлення втрачає будь-які властивості повідомлення що було отримане на вході. Це призведе до руйнування деяких потоків, наприклад потік HTTP In/Response вимагає щоб властивості msg.req і msg.res повинні бути збережені від кінця до кінця. Загалом, вузли Function *повинні* повернути об'єкт-повідомлення, вносячи будь-які зміни до його властивостей.

Відправка повідомлень на декілька виходів (Sending to multiple outputs)

Діалог редагування Function дозволяє змінити кількість виходів. Якщо є більше одного виходу, для відправки на виходи функцією може бути повернутий масив повідомлень.

Це полегшує написання функції, яка відправляє повідомлення на різні виходи в залежності від певного стану. Наприклад, ця функція буде надсилати що-небудь на тему (topic) banana до другого виходу, а не до першого

```
if (msg.topic === "banana") {
    return [ null, msg ];
} else {
    return [ msg, null ];
}
```

У наступному прикладі, оригінальне вхідне повідомлення передається на перший вихід, а на другий вихід передається повідомлення, яке містить довжину рядку payload:

```
var newMsg = { payload: msg.payload.length };
return [msg, newMsg];
```

Відправка кількох повідомлень (Sending multiple messages)

Функція може повертати послідовність декількох повідомлень на виході, повернувши масив повідомлень у середині іншого масиву. Наступні за ним вузли отримуватимуть повідомлення один за одним у тому порядку, в якому вони були у повернутому масиві цього вузла.

У наступному прикладі, послідовність повідомлень `msg1`, `msg2`, `msg3` буде відправлена на перший вихід, а повідомлення `msg4` буде відправлено на другий вихід.

```
var msg1 = { payload:"first out of output 1" };
var msg2 = { payload:"second out of output 1" };
var msg3 = { payload:"third out of output 1" };
var msg4 = { payload:"only message from output 2" };
return [ [ msg1, msg2, msg3 ], msg4 ];
```

У наступному прикладі розбивається отримане корисне навантаження (payload) на окремі слова і повертається повідомлення, що вміщує кожне слово.

```
var outputMsgs = [];
var words = msg.payload.split(" ");
for (var w in words) {
    outputMsgs.push({payload:words[w]});
}
return [ outputMsgs ];
```

Відправка повідомлень асинхронно (Sending messages asynchronously)

Якщо функція повинна виконувати асинхронну дію (що не відразу повертає результат а потребує функцію зворотного виклику) що приймає участь у формуванні повідомлення, вона не може повернути повідомлення по завершенню обробки Function.

Замість цього повинна використовуватися функція передачі повідомлень `node.send()`. Наприклад:

```
doSomeAsyncWork(msg, function(result) {
    node.send({payload:result});
});
return;
```

Якщо ви використовуєте асинхронний код зворотного виклику у своїх функціях, коли потік повторно розгортається, вам знадобиться прибирати будь-які непотрібні запити або закривати будь-які з'єднання. Ви можете зробити це, додавши обробник подій `close`

```
node.on('close', function() {
    // tidy up any async code here - shutdown connections and so on.
});
```

Ведення журналу подій (Logging events)

Якщо вузол повинен записувати щось в консоль, він може використовувати одну з наступних функцій:

```
node.log("Something happened");
node.warn("Something happened you should know about");
node.error("Oh no, something bad happened");
```

Повідомлення `warn` і `error` також надсилаються на вкладку налагодження редактора потоку.

Для більш тонкорівневих подій також доступні `node.trace()` і `node.debug()` . Якщо для цих рівнів не налаштоване ведення журналу, вони відображатися не будуть.

Обробка помилок (Handling errors)

Якщо функція виявляє помилку, вона припиняє поточний потік і нічого не повертає. Щоб запустити вузол Catch на тій самій вкладці, функція повинна викликати `node.error` з оригінальним повідомленням в якості другого аргументу:

```
node.error("hit an error", msg);
```

Зберігання даних (Storing data (context))

Крім об'єкту `msg` функція також може зберігати дані в контекстному сховищі.

Більш детальна інформація про контекст в рамках Node-RED доступна тут.

У вузлі Function є три заздалегідь визначені змінні, які можна використовувати для доступу до контексту:

- `context` - локальний контекст вузла
- `flow` - контекст потоку
- `global` - глобальний контекст

Наступні приклади використовують контекст потоку `flow` , але аналогічно цей приклад застосовний і до `context`, і до `global`.

Примітка: ці заздалегідь означені змінні є особливостями вузла Function. Якщо ви створюєте спеціальний вузол, прочитайте Creating Nodes guide для ознайомлення з тим, як отримати доступ до контексту.

Існує два режими доступу до контексту: синхронний та асинхронний. Вбудовані контекстні сховища забезпечують обидва режими. Деякі сховища можуть надавати лише асинхронний доступ і вибивають помилку, якщо до них доступаються синхронно.

Щоб отримати значення з контексту:

```
var myCount = flow.get("count");
```

Щоб встановити значення:

```
flow.set("count", 123);
```

У наведеному нижче прикладі зберігається лічильник кількості викликів функції:

```
// ініціалізація змінної лічильника 'count' в 0, якщо її немає в контексті вузла
var count = context.get('count')||0;
count += 1;
// збереження значення store в контексті вузла під іменем 'count'
context.set('count',count);
// зробити це частиною вихідного об'єкту  msg
msg.count = count;
return msg;
```

Отримати/Встановити кілька значень

Node-RED може отримати або встановити декілька значень за один раз:

```
// Node-RED 0.19 або пізніше
var values = flow.get(["count", "colour", "temperature"]);
// values[0] це значення 'count'
// values[1] це значення 'colour'
// values[2] це значення 'temperature'

// Node-RED 0.19 або пізніше
flow.set(["count", "colour", "temperature"], [123, "red", "12.5"]);
```

У цьому випадку для будь-яких відсутніх значень буде встановлено значення `null`.

Асинхронний доступ до контексту

Якщо для контекстного сховища потрібен асинхронний доступ, то функції `git` та `set` вимагають додаткового параметра функції зворотного виклику.

```
// отримати одне значення
flow.get("count", function(err, myCount) { ... });

// отримати кілька значень
flow.get(["count", "colour"], function(err, count, colour) { ... })

// встановити одне значення
flow.set("count", 123, function(err) { ... })

// встановити кілька значень
flow.set(["count", "colour", [123, "red"], function(err) { ... })
```

Перший аргумент у функції зворотного виклику, err, встановлюється лише тоді, коли при доступі до контексту виникла помилка.

Приклад асинхронної версії:

```
context.get('count', function(err, count) {
    if (err) {
        node.error(err, msg);
    } else {
        // ініціалізація змінної лічильника 'count' в 0, якщо її
        // немає в контексті вузла
        count = count || 0;
        count += 1;
        // збереження значення store в контексті вузла під іменем 'count'
        context.set('count',count, function(err) {
            if (err) {
                node.error(err, msg);
            } else {
                // зробити це частиною вихідного об'єкту  msg
                msg.count = count;
                // відправити повідомлення
                node.send(msg);
            }
        });
    }
});
```

Кілька контекстних сховищ

З версії 0.19 можна налаштувати кілька контекстних сховищ. Наприклад, можуть бути використані два типа сховищ, що базуються на memory і file .

Функції роботи з контекстом get/set приймають необов'язковий параметр (у прикладі це storeName) для ідентифікації сховища для використання.

```
// отримати значення синхронно - sync
var myCount = flow.get("count", storeName);

// отримати значення асинхронно - async
flow.get("count", storeName, function(err, myCount) { ... });

// встановити значення синхронно - sync
flow.set("count", 123, storeName);

// встановити значення асинхронно - async
flow.set("count", 123, storeName, function(err) { ... })
```

Глобальний контекст

Коли Node-RED запускаються глобальний контекст може бути попередньо заповнений об'єктами. Це означено в основному файлі *settings.js* властивістю `functionGlobalContext` .

Це може бути використано для Завантаження додаткових модулів в межах вузла Function.

Додавання статусу (Adding status)

Вузол Function також може забезпечити власне оформлення статусу таким же чином, як і інші вузли. Щоб встановити статус потрібно викликати `node.status` Наприклад:

```
node.status({fill:"red",shape:"ring",text:"disconnected"});
node.status({fill:"green",shape:"dot",text:"connected"});
node.status({text:"Just text status"});
node.status({});    // для очищення статусу
```

Докладніше про прийняті параметри дивитись Node Status documentation

Тоді будь-які оновлення статусу можуть також потрапляти до вузла Status.

Завантаження додаткових модулів (Loading additional modules)

У вузлі Function додаткові модулі не можуть бути завантажені безпосередньо. Вони повинні бути завантажені у вашому файлі *settings.js* і добавлені до властивості `functionGlobalContext`.

Наприклад, вбудований модуль `os` може бути доступним для всіх функцій, додавши його до наступного файлу *settings.js*

```
functionGlobalContext: {
    osModule:require('os')
}
```

після чого на модуль можна посилатися в межах функції як `global.get('osModule')`.

Модулі, завантажені з вашого файлу налаштувань, повинні бути встановлені в тому ж каталозі, що і файл налаштувань. Для більшості користувачів цей каталог буде користувачем за замовчуванням - `~/.node-red`:

```
cd ~/.node-red
npm install name_of_3rd_party_module
```

Довідник API (API Reference)

Наступні об'єкти доступні в межах вузла Function.

node

- `node.id` : ідентифікатор вузла Function - *додано в 0,19*
- `node.name` : назва функціонального вузла - *додано в 0,19*
- `node.log(..)` : запис повідомлення
- `node.warn(..)` : log a warning message
- `node.error(..)` : log an error message
- `node.debug(..)` : log a debug message
- `node.trace(..)` : log a trace message
- `node.on(..)` : register an event handler
- `node.status(..)` : update the node status
- `node.send(..)` : send a message

context

- `context.get(..)` : отримати властивість контексту вузлу
- `context.set(..)` : встановити властивість контексту вузлу
- `context.keys(..)` : повертає список всіх ключів властивостей контексту вузлу
- `context.flow` : такий же як `flow`
- `context.global` : такий же як `global`

flow

- `flow.get(..)` : отримати властивість контексту потоку
- `flow.set(..)` : встановити властивість контексту потоку
- `flow.keys(..)` : повертає список усіх ключів властивостей контексту потоку

global

- `global.get(..)` : отримати властивість глобального контекстну
- `global.set(..)` : встановити властивість глобального контексту
- `global.keys(..)` : повернути список усіх ключів властивостей глобального контексту

RED

- `RED.util.cloneMessage(..)` : безпечно клонує об'єкт повідомлення, щоб його можна було використовувати повторно

Інші модулі та функції

Вузол Function також робить доступними наступні модулі та функції:

- `Buffer` - модуль Node.js `Buffer`
- `console` - модуль Node.js `console` (`node.log` є наперед визначеним методом ведення журналу)

- `util` - модуль Node.js `util`
- `setTimeout/clearTimeout` - функція тайм-ауту javascript.
- `setInterval/clearInterval` - функції інтервалу javascript.

Примітка: вузол Function автоматично очищає будь-які прострочені тайм-аути або таймери інтервалів, коли він зупиняється або повторно розгортається.

Робота з повідомленнями (Working with messages)

Потік Node-RED працює, передаючи повідомлення між вузлами. Повідомлення є простими об'єктами JavaScript, які можуть мати будь-який набір властивостей.

Повідомлення звичайно мають властивість payload це властивість за умовчанням, з яким працюватиме більшість вузлів

Node-RED також додає властивість, що називається `msgid` - це ідентифікатор для повідомлення, яке може використовуватися для відстеження його проходження потоком

```
{
"_msgid": "12345",
    "payload": "..."
}
```

Значенням властивості може бути будь-який дійсний тип JavaScript, наприклад

- Boolean - `true`, `false`
- Number – наприклад `0`, `123.4`
- String - `"hello"`
- Array - `[1,2,3,4]`
- Object - `{ "a": 1, "b": 2}`
- Null

Докладніше про типи JavaScript

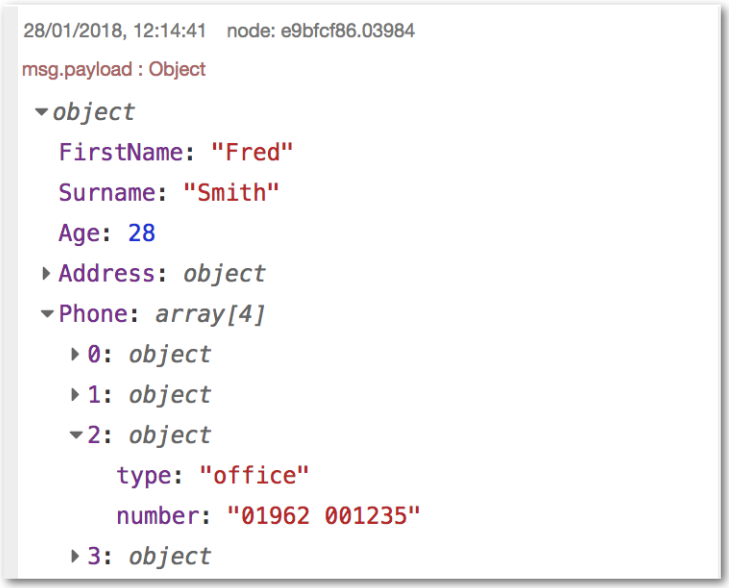
Розуміння структури повідомлень (Understanding the structure of a message)

Найпростіший спосіб зрозуміти структуру повідомлення - передати його в вузол Debug і переглянути його на бічній панелі Debug.

За замовчуванням на вузлі Debug відображатиметься властивість `msg.payload` , але може бути налаштована для відображення будь-яка інша властивість або все повідомлення цілком.

При відображенні масиву або об'єкту бічна панель забезпечує структурований вигляд, який може використовуватися для вивчення повідомлення.

- Угорі він показує ім'я властивості, яке було передано. Тут за замовчуванням використано `msg.payload`
- Поруч із назвою є назва типу властивості - `Object`, `String`, `Array` ін..
- Потім він показує вміст властивості. Для масивів і об'єктів властивість розпадається на лінії. Клацаючи по ньому, властивість розгорнеться, щоб показати більш детальну інформацію.



Коли ви наводите курсор миші на будь-який елемент, праворуч з'являється набір кнопок:

- :  копіює шлях до обраного елемента у буфер обміну. У цьому прикладі він буде копіювати `payload.Phone[2].type`. Це дозволяє швидко визначити, як отримати доступ до властивості на вузлі Change або Function.

- :  копіює значення елемента у буфер обміну як рядок JSON. Зауважте, що бічна панель обробляє масиви та буфери певної довжини. Копіюючи значення такої властивості, буде скопійована урізана версія.
- :  вибирає елемент таким чином, що він завжди відображається. Коли одне повідомлення одержується з того ж вузла відлагодження, його автоматично розширюють, щоб показати всі закріплені елементи.

Робота з JSON (Working with JSON)

JSON, (JavaScript Object Notation) - це стандартний спосіб подання об'єкта JavaScript як рядка. Він часто використовується у веб-API для повернення даних.

Якщо властивість повідомлення містить рядок JSON, то перед тим, як мати доступ до його властивостей, його слід перетворити до еквівалентного об'єкта JavaScript. Щоб визначити, чи властивість вміщує String чи Object, може бути використаний вузол Debug.

Node-RED забезпечує вузол `JSON` для здійснення цього перетворення.

Зміна властивостей повідомлення (Changing message properties)

Спільне завдання в потоці - змінити властивості повідомлення, коли воно проходить між вузлами. Наприклад, результатом роботи вузлу `HTTP Request` може бути об'єкт з багатьма властивостями, з яких потрібні лише деякі.

Для модифікації повідомлення використовують два основні вузли – Function та Change.

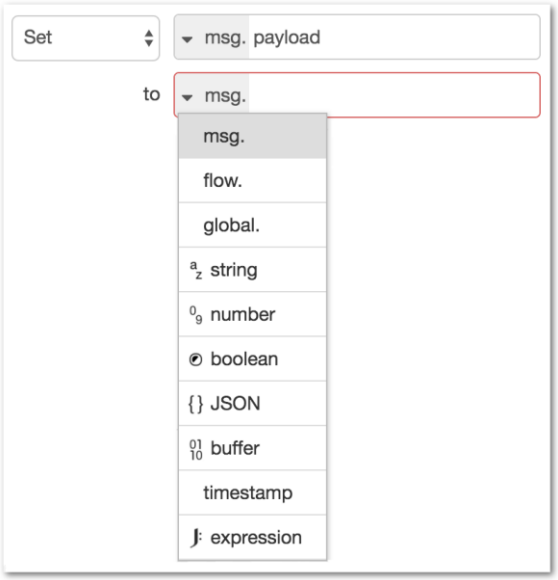
Вузол Function дозволяє запускати будь-який код JavaScript для обробки повідомлення. Це дає вам повну гнучкість в тому, що ви робите з повідомленням, але вимагає знайомства з JavaScript і не є необхідним для багатьох простих випадків. Більше інформації про написання функцій доступно тут.

Вузол Change забезпечує багато функціональних можливостей без необхідності писати код JavaScript. Він не тільки може змінювати властивості повідомлення, але також може отримати доступ до контексту потоку або глобального контексту.

Він забезпечує чотири основні операції:

- `Set` - встановлення значення властивості,
- `Change` - пошук та заміна частини властивості типу String
- `Delete` – видалення властивості
- `Move` – переміщення властивості.

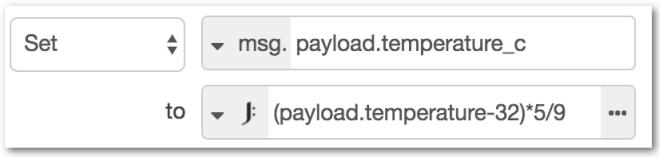
Для операції set ви спочатку означуєте, яку властивість ви хочете встановити, тоді значення, яке ви хочете мати. Це значення може бути або жорстко закріплене, наприклад, рядок чи число, або це може бути взяте з іншого повідомлення або властивості контексту потоку/глобального. Він також підтримує використання JSONata для розрахунку нового значення.



Наприклад, за допомогою можливості вузлів Debug визначати шлях повідомлень, ви можете вставити шлях прямо в поле ‘to’, з вибраного списку `msg`. Тоді буде встановлено `msg.payload` в значення `msg.payload.Phone[2].type`.

For example, using the Debug node’s ability to determine a message element’s path, you can paste the path straight into the ‘to’ field, with `msg.` selected from the list. That will then set `msg.payload` to the value of `msg.payload.Phone[2].type`

Інший приклад виразу JSONata полягає в перетворенні температури, що зберігається в `msg.payload.temperature` з одиниць Фаренгейта до Цельсія та збереження результату у новій властивості повідомлення `msg.payload.temperature_c`.




```
{
    "payload": {
        "temperature": 90,
        "temperature_c": 32.22222
    }
}
```

Зауважте, що вирази *JSONata виглядають так само, як JavaScript, але мають деякі ключові відмінності. Див* jsonata.org сайт для додаткової інформації.

Послідовності повідомлень (Message sequences)

Послідовність повідомлень - це впорядкована серія повідомлень, які певним чином пов'язані між собою. Наприклад, вузол Split може перетворювати одне повідомлення, яке є масивом `payload` , у послідовність повідомлень, де кожне повідомлення містить `payload` що відповідає одному з елементів масиву.

Розуміння `msg.parts` (Understanding msg.parts)

Кожне повідомлення в послідовності має властивість `msg.parts`. Це об'єкт, який містить інформацію про те, як повідомлення входить у послідовність. Він має такі властивості:

`msg.parts.id` - унікальний ідентифікатор в послідовності
`msg.parts.index` - позиція повідомлення в межах послідовності
`msg.parts.count` - якщо відомо, загальна кількість повідомлень у послідовності

Примітка: масив parts може містити додаткові метадані про послідовність. Наприклад, вузол split також надає інформацію, яка може бути використана вузлом join для повторного зібрання послідовності. Див документацію для вузла split .

Робота з послідовностями (Working with sequences)

Є кілька основних вузлів, які можуть працювати з послідовностями повідомлень

Split

Перетворює одне повідомлення в послідовність повідомлень.

Конкретна поведінка вузла залежить від типу `msg.payload`:

- String/Buffer - повідомлення розділяється за допомогою заданого символу (за замовчуванням: `\ n`), буферна послідовність або фіксована довжина. The message is split using the specified character (default: `\n`), buffer sequence or into fixed lengths.
- Array - повідомлення розділяється на окремі елементи масиву чи масиви фіксованої довжини
- Object - повідомлення надсилається для кожної пари об'єкта ключ/значення.

Join

Перетворює послідовність повідомлень у єдине повідомлення.

Вузол забезпечує три режими роботи:

- Automatic - спроба зробити зворотну дію, проведену попереднім вузлом `Split`
- Manual - дозволяє точніше керувати, як слід об’єднати послідовність
- Reduce - дозволяє виконувати вираз JSONata для кожного повідомлення в послідовності, а накопичений результат використати для створення одного повідомлення.

Sort

Сортує послідовність повідомлень на основі значення властивості або результату вираження JSONata.

Batch

Створює нові послідовності повідомлень з отриманих.

Вузол забезпечує три режими роботи:

- Number of messages - групування повідомлень в послідовності заданої довжини. Параметр overlap (перекривання) вказує, скільки повідомлень в кінці однієї послідовності слід повторити на початку наступної послідовності.
- Time interval - групові повідомлення, що надходять у вказаний інтервал. Якщо протягом вказаного інтервалу не надходить повідомлення, вузол може додатково надіслати порожнє повідомлення.
- Concatenate Sequences - створює послідовність повідомлень шляхом об'єднання вхідних послідовностей. Кожна послідовність повинна мати властивість msg.topic для визначення групи. Вузол налаштовується на список значень тем для ідентифікації порядку конкатенації послідовностей.

Робота з контекстом (Working with context)

Що таке контекст

Node-RED забезпечує спосіб зберігання інформації, яка може бути розподілена між різними вузлами, без використання повідомлень, що проходять через потоки. Це називається ‘context’.

Область видимості контексту

Область видимості конкретного значення контексту визначається від того, хто його надає. Існує три рівні контексту:

- Node - видимий тільки для вузла, який встановлює значення
- Flow - видимий для всіх вузлів на одному потоці (або вкладки у редакторі)
- Global - видимий для всіх вузлів

Вибір області видимості для будь-якого конкретного значення залежить від того, як воно використовується.

Якщо для доступу до значення потрібно лише один вузол, наприклад, вузол Function, то рівень контексту Node достатній.

Найчастіше контекст дозволяє розділяти певний стан між кількома вузлами. Наприклад, датчик може регулярно публікувати нові значення в одному потоці, і ви хочете створити окремий потік HTTP, щоб повернути останнє значення. Зберігаючи зчитане значення датчику в контексті, він потім доступний для повернення потоку HTTP.

Глобальний контекст Global може бути попередньо зконфігурований значеннями з використанням властивості `functionGlobalContext` в файлі налаштувань.

Примітка : для вузлів з під-потоку, контекст 'flow' є областю під-потоку. Вузли не мають доступу до контексту потоку, що містить вузол екземпляра під-потоку.

Контекстні сховища

За замовчуванням контекст зберігається лише в пам'яті. Це означає, що його вміст очищується, коли Node-RED перезавантажується. З випуском 0,19 можна налаштувати Node-RED для збереження контекстних даних, щоб він став доступним і після перезавантаження.

Властивість `contextStorage` в settings.js можна використовувати для налаштування того, як будуть зберігатися контекстні дані.

Наприклад, щоб увімкнути сховище на базі файлів, можна використовувати наступні параметри:

```
contextStorage: {
    default: {
        module: "localfilesystem"
    }
}
```

Node-RED забезпечує два вбудованих модуля сховища: `memory` і `localfilesystem`. Також можливо створити власні плагіни сховища.

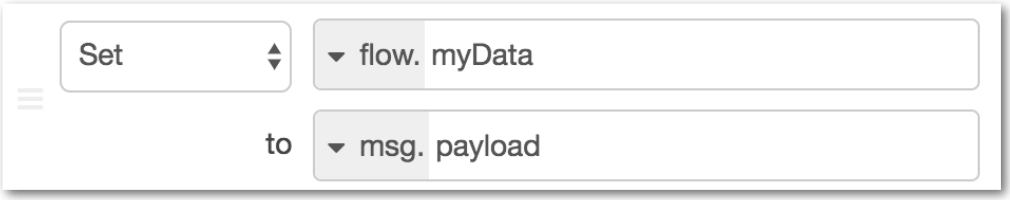
Повна інформація про вбудовані модулі та способи створення користувацьких модулів доступна на api pages.

Сховища можуть забезпечити як синхронний, так і асинхронний доступ. Синхронний доступ означає, що виклик для отримання даних з сховища негайно повертає значення. Асинхронний доступ означає, що функція виклику для отримання даних також повинен містити функцію зворотного виклику, яка викликається після того, як значення стало доступним.

Вбудовані сховища `memory` і `file` обидва пропонують синхронний доступ. Це означає, що існуючі (попередні до версії 0,19) потоки можуть використовувати ці сховища без будь-яких змін..

Використання контексту в потоці

Найпростішим способом встановити значення контексту є використання вузла Change. Наприклад, наступне правило Change node зберігатиме значення `msg.payload` в контексті потоку (`flow`) під ключем `myData`.



Різні вузли можуть безпосередньо отримати доступ до контексту. Наприклад, вузол Inject може бути налаштований для введення значення в контекст, а вузол Switch може маршрутизувати повідомлення на основі значення, збереженого в контексті.

Використання контексту у вузлі Function

Writing Functions guide описує, як використовувати контекст у вузлі Function.

Використання контексту в користувацькому вузлі

Creating Nodes guide описує, як використовувати контекст у користувацькому вузлі.

Запуск Node-RED (Running Node-RED)

Цей розділ є частково проробленим .

Конфігурація (Configuration)

Наведені нижче властивості можуть бути використані для налаштування Node-RED.

Якщо запустити Node-RED як автономну програму, ці властивості читаються з файлу `settings.js` Розташування цього файлу визначається в наступному порядку:

- встановленого за допомогою `--settings|-s` аргументу командного рядка
- в каталозі користувача, якщо це було вказано в `--userDir|-u` аргументі командного рядка
- в каталозі користувача за замовчуванням: `$HOME/.node-red/settings.js`
- в каталозі, де встановлений node-red

Node-RED включає за замовчуванням файл `settings.js` , який буде використовуватися за відсутності користувацького файлу налаштувань. Він також може використовуватися як відправна точка для створення власного файлу налаштувань. Це можна побачити на GitHub тут.

Приклад : Файл settings.js експортує об’єкт *JavaScript* Щоб налаштувати Node-RED, ви повинні зрозуміти, як змінити об'єкт JavaScript, додавши нові або змінюючи існуючі пари ключів/значень.

Якщо Node-RED запускається, як вбудована, вона передає виклик `RED.init()`. Однак при запуску в цьому режимі деякі властивості ігноруються і залишаються в додатку для вбудовування для реалізації.

Конфігурація середовища виконання

flowFile - файл, який використовується для зберігання потоків. За замовчанням: `flows_<hostname>.json`

userDir - каталог для зберігання всіх користувацьких даних, таких як поточні та облікові файли та всі бібліотечні дані. За замовчанням: `$HOME/.node-red`

nodesDir - каталог для пошуку додаткових встановлених вузлів. Node-RED шукає каталог nodes в каталозі *userdir*. Ця властивість вказує де шукати додатковий каталог, щоб вузли могли бути встановлені за межами структури встановлення Node-RED. За замовченням `$HOME/.node-red/nodes`

uiHost - інтерфейс для прослуховування з'єднань. За замовченням: 0.0.0.0 – *всі інтерфейси IPv4*

Тільки для Standalone.

uiPort - порт використовується для обслуговування редактора інтерфейсу. За замовченням :1880.

Тільки для Standalone.

httpAdminRoot - корінь url для інтерфейсу редактора UI. Якщо встановлено false, усі кінцеві точки адміністратора вимкнені. Це включає як кінцеві точки API, так і інтерфейс редактора. Щоб вимкнути лише інтерфейс редактора, див. властивість disable edditor нижче. За замовченням: /

httpAdminAuth – застаріле, див `adminAuth`.

дозволяє HTTP Basic Authentication в інтерфейсі редактора:

```
httpAdminAuth: {user:"nol", pass:"5f4dcc3b5aa765d61d8327deb882cf99"}
```

Властивість `pass` є md5 фактичного пароля. Для генерації хешу може бути використана наступна команда:

```
node -e "console.log(require('crypto').createHash('md5').update('YOUR PASSWORD HERE','utf8').digest('hex'))"
```

Тільки для Standalone.

httpNodeRoot - кореневий URL для вузлів, які надають кінцеві точки HTTP. Якщо встановлено false , всі кінцеві точки HTTP на вузлах вимикаються. За замовченням: /

httpNodeAuth - дозволяє базову автентифікацію HTTP. Подивіться httpAdminAuth для формату.

httpRoot - це встановлює кореневий URL для кінцевих точок адміністратора та вузлів. Це перевизначає значення, встановлені `httpAdminRoot` і `httpNodeRoot`.

https - дозволяє https, з вказаним об'єктом параметрів, як визначено тут.

Тільки для Standalone.

disableEditor - якщо встановлено true заважає інтерфейсу редактора працювати під час виконання. Кінцеві точки адміністратора api залишаються активними. За замовченням: `false`.

httpStatic - локальний каталог, в якому можна показувати статичний веб-вміст. Цей вміст подано з URL-адреси верхнього рівня, / . Коли ця властивість використовується, httpAdminRoot також повинен використовуватися для створення користувальницького інтерфейсу користувача за іншим шляхом /.

Тільки для Standalone.

httpStaticAuth - включена основна автентифікація HTTP на статичному вмісті. Див. httpAdminAuth для формату.

httpNodeCors - дозволяє розподіляти ресурси між джерелами для вузлів, які надають кінцеві точки HTTP, як це визначено тут

httpNodeMiddleware - функція HTTP проміжного програмного забезпечення, додана до всіх HTTP- In вузлів. Це дозволяє виконувати будь-яку індивідуальну обробку, наприклад

аутентифікацію, для вузлів. Формат функції проміжного програмного забезпечення задокументований тут.

```
httpNodeMiddleware: function(req,res,next) {
    // Perform any processing on the request.
    // Be sure to call next() if the request should be passed
    // to the relevant HTTP In node.
}
```

logging - в даний час підтримується лише консольний журнал. Можна вказати різні рівні ведення журналу. Опції:

- **fatal** - only those errors which make the application unusable should be recorded
- **error** - record errors which are deemed fatal for a particular request + fatal errors
- **warn** - record problems which are non fatal + errors + fatal errors
- **info** - record information about the general running of the application + warn + error + fatal errors
- **debug** - record information which is more verbose than info + info + warn + error + fatal errors
- **trace** - record very detailed logging + debug + info + warn + error + fatal errors

currently only console logging is supported. Various levels of logging can be specified. Options are:

- **fatal** - only those errors which make the application unusable should be recorded
- **error** - record errors which are deemed fatal for a particular request + fatal errors
- **warn** - record problems which are non fatal + errors + fatal errors
- **info** - record information about the general running of the application + warn + error + fatal errors
- **debug** - record information which is more verbose than info + info + warn + error + fatal errors
- **trace** - record very detailed logging + debug + info + warn + error + fatal errors

Стандартний рівень є `info`. Для вбудованих пристроїв із обмеженою флеш-пам'яттю ви можете встановити його як `fatal` для мінімізації запису на“disk”.

Конфігурація середовища розробки (редактора)

adminAuth - забезпечує безпеку на рівні користувача в редакторі та API адміністратора. Див. security для отримання додаткової інформації.

paletteCategories - визначає порядок категорій у палітрі. Якщо категорія вузла відсутня в списку, ця категорія буде додана до кінця палітри. Якщо не встановлено, використовується такий порядок за замовчуванням:

```
['subflows', 'input', 'output', 'function', 'social', 'storage', 'analysis',
'advanced'],
```

Примітка: поки користувач не створить під-поток, категорія під-потоків буде порожньою і не буде видною в палітрі.

Теми редактора

Тему редактора можна змінити, використовуючи наступний об'єкт параметрів. Всі частини необов'язкові.

```
editorTheme: {
```

```
    page: {
        title: "Node-RED",
        favicon: "/absolute/path/to/theme/icon",
        css: "/absolute/path/to/custom/css/file"
    },
    header: {
        title: "Node-RED",
        image: "/absolute/path/to/header/image", // or null to remove image
        url: "http://nodered.org" // optional url to make the header text/image a link to
this url
    },
    deployButton: {
        type:"simple",
        label:"Save",
        icon: "/absolute/path/to/deploy/button/image" // or null to remove image
    },
    menu: { // Hide unwanted menu items by id. see editor/js/main.js:loadEditor for
complete list
        "menu-item-import-library": false,
        "menu-item-export-library": false,
        "menu-item-keyboard-shortcuts": false,
        "menu-item-help": {
            label: "Alternative Help Link Text",
            url: "http://example.com"
        }
    },
    userMenu: false, // Hide the user-menu even if adminAuth is enabled
    login: {
        image: "/absolute/path/to/login/page/big/image" // a 256x256 image
    }
},
```

Конфігурація Dashboard

ui - можна означити шлях на домашню сторінку для додаткових вузлів Node-RED-Dashboard Це стосується будь-якого вже означеного **httpNodeRoot**

ui : { path: “mydashboard” },

Конфігурація вузла

Будь-який тип вузла може означати власні параметри, які будуть надані у файлі.

functionGlobalContext

Function Nodes - це сукупність об'єктів, що підключаються до функції глобального контексту. Наприклад,

```
functionGlobalContext: { osModule:require('os') }
```

можна отримати доступ до функції вузла як:

```
var myos = global.get('osModule');
```

Примітка : До Node-RED v0.13 документованим способом використання глобального контексту було доступ до нього як суб-властивіст `context`:

Note : Prior to Node-RED v0.13, the documented way to use global context was to access it as a sub-property of `context`:

```
context.global.foo = "bar";
 var osModule = context.global.osModule;
```

Цей метод все ще підтримується, але застарілий на користь `global.get` / `global.set` функції. Це передбачає можливість зберігати контекстні дані у майбутньому випуску.
This method is still supported, but deprecated in favour of the `global.get`/`global.set`functions. This is in anticipation of being able to persist the context data in a future release.
debugMaxLength
Debug Nodes- максимальна довжина символів будь-якого повідомлення, надісланого на вкладку бічній панелі відладки. За замовчуванням: 1000
Debug Nodes - the maximum length, in characters, of any message sent to the debug sidebar tab. Default: 1000

mqttReconnectTime
MQTT Nodes - якщо з'єднання втрачено, скільки часу потрібно чекати в мілісекундах, перш ніж намагатись знову підключитися. За замовчуванням: 5000
MQTT Nodes - if the connection is lost, how long to wait, in milliseconds, before attempting to reconnect. Default: 5000

serialReconnectTime
Serial Nodes - скільки часу потрібно чекати в мілісекундах, перш ніж намагатись відкрити послідовний порт. За замовчуванням: 5000
Serial Nodes - how long to wait, in milliseconds, before attempting to reopen a serial port. Default: 5000

socketReconnectTime
TCP Nodes - скільки часу потрібно чекати в мілісекундах, перш ніж намагатись знову підключитися. За умовчанням: 10000
TCP Nodes - how long to wait, in milliseconds, before attempting to reconnect. Default: 10000

socketTimeout
TCP Nodes - скільки часу потрібно зачекати, за мілісекунди, перед тим, як закінчити розетку. По замовчуванню: 120000

TCP Nodes - how long to wait, in milliseconds, before timing out a socket. Default: 120000

Безпека (Security)

За замовчуванням редактор Node-RED не захищений - кожен, хто може отримати доступ до IP-адреси і порт, на якому він працює, може отримати доступ до редактора та впроваджувати зміни. Це підходить лише для роботи в надійній мережі.

У цьому розділі описано, як можна захистити Node-RED. Безпека поділена на дві частини:

- API редактору і адміністратору
  - автентифікація на основі логіну і паролю
    - Створення хеш-пароля
  - Автентифікація на базі OAuth / OpenID
  - Користувач за змовчуванням
  - Права користувачів
  - Термін дії Token
  - Доступ адміністратора API

API редактору та адміністратору

Редактор та API адміністратору підтримує два типи автентифікації:

- автентифікація на основі введення імені користувача та пароля
- *починаючи з Node-RED 0.17:* автентифікація що спирається на будь-якого постачальника OAuth/OpenID такого як Twitter чи GitHub

Автентифікація на основі логіну і паролю (Username/password based authentication)

Щоб увімкнути аутентифікацію користувача в API редактору та адміністратору, необхідно додати до вашого файлу `settings.js` наступне:

```
adminAuth: {
    type: "credentials",
    users: [{
        username: "admin",
        password: "$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
        permissions: "*"
    }]
}
```

Властивість `users` – це масив об'єктів користувача. Це дозволяє визначити декілька користувачів, кожен з яких може мати різні дозволи.

Цей приклад конфігурації означує одного користувача, який викликається `admin` який має дозвіл робити все в редакторі та має пароль `password`. Зверніть увагу, що пароль надійно змінено за допомогою алгоритму bcrypt.

Примітка : у попередніх версіях Node-RED, для активації редакторі HTTP Basic Authentication може використовуватися налаштування `httpAdminAuth` . Цей параметр застарілий і не повинен використовуватися

Створення хеш-пароля

Щоб створити відповідний хеш-пароль, ви можете скористатись інструментом командного рядка `node-red-admin`. Інструкції з встановлення інструменту доступні тут.

```
node-red-admin hash-pw
```

Утиліта запропонує вам пароль, який ви хочете використовувати, а потім видасть хеш, який можна скопіювати у файл налаштувань.

Альтернативний шлях - запустити таку команду з каталогу встановлення Node-RED:

```
node -e "console.log(require('bcryptjs').hashSync(process.argv[1], 8));" your-password-here
```

Автентифікація на базі OAuth/OpenID

Починаючи з Node-RED 0.17

Щоб використовувати зовнішнє джерело автентифікації, Node-RED може використовувати широкий спектр стратегій, що надаються в Passport.

Модулі аутентифікації Node-RED доступні як для Twitter так і для GitHub. Вони складають окремі деталі стратегії, щоб полегшити їх використання. Але вони також можуть бути використані як шаблон для аутентифікації з іншими подібними стратегіями.

Наступний приклад показує, як налаштувати аутентифікацію на Twitter *без* використання авторизаційного модуля, який ми надаємо.

```
adminAuth: {
    type:"strategy",
    strategy: {
        name: "twitter",
        label: 'Sign in with Twitter',
        icon:"fa-twitter",
        strategy: require("passport-twitter").Strategy,
        options: {
            consumerKey: TWITTER_APP_CONSUMER_KEY,
            consumerSecret: TWITTER_APP_CONSUMER_SECRET,
            callbackURL: "http://example.com/auth/strategy/callback"
        },
        verify: function(token, tokenSecret, profile, done) {
            done(null, profile);
        }
    },
    users: [
       { username: "knolleary",permissions: ["*"]}
    ]
};
```

Властивість `strategy` приймає такі варіанти:

- `name` - назва паспортної стратегії, що використовується
- `strategy` - модуль паспортної стратегії
- `label`/`icon` - використовується на сторінці входу. `icon` може бути будь-яким значком FontAwesome.
- options - об'єкт опцій, який передається до паспортної стратегії, коли вона створюється. Зверніться до власної документації стратегії, для чого вона потрібна. Нижче дивіться на вузол callbackURL.
- verify - функція перевірки, яка використовується стратегією. Потрібно викликати done з використанням профілю користувача як другого аргументу, якщо користувач є дійсним. Очікується, що це буде властивість username яка використовується для перевірки зі списку дійсних користувачів. Паспорт намагається стандартизувати об'єкт профілю користувача, тому більшість стратегій забезпечують цю властивість.

callbackURL використовується як стратегія, коли постачальник автентифікації переспрямовує на наступну спробу авторизації. Це має бути URL-адреса редактора Node-RED з /auth/strategy/callback доданим шляхом. Наприклад, якщо ви маєте доступ до редактора на http://localhost:1880, ви б могли використовувати http://localhost:1880/auth/strategy/callback.

Користувач за замовчуванням

Наведений вище приклад не дозволить будь-кому отримати доступ до редактора, якщо він не ввійшов у систему.

У деяких випадках бажано дозволити кожному певний рівень доступу. Як правило, це дасть доступ лише для читання в редакторі. Щоб це зробити до налаштувань adminAuth може бути додана властивість default для визначення користувача за умовчанням:

```
adminAuth: {
    type: "credentials",
    users: [ /* list of users */ ],
    default: {
        permissions: "read"
    }
}
```

Права користувачів

Перед Node-RED 0.14 користувачі можуть мати один із двох дозволів:

- `*` - повний доступ
- `read` – тільки читання

Починаючи з Node-RED 0.14 дозволи можуть бути набагато більш тонкорівневими і підтримувати їх властивість може як однин рядок, так і масив, що містить кілька дозволів.

Кожен метод Admin API означує, який рівень дозволу потрібен для доступу до нього. Модель дозволу базується на основі ресурсів. Наприклад, щоб отримати поточну конфігурацію, користувачеві буде потрібно дозвіл flows.read. Але для оновлення потоків їм знадобиться дозвіл flows.write.

Термін дії Token

За замовчуванням маркери доступу закінчуються через 7 днів після їх створення. Наразі Node-RED не підтримує оновлення токену більше ніж цей час.

Час вичерпання може бути налаштований шляхом встановлення властивості sessionExpiryTime в налаштуваннях adminAuth. Він вказується наскільки токен дійсний у секундах. Наприклад, щоб встановити термін дії токенів на 1 день:

```
adminAuth: {
    sessionExpiryTime: 86400,
    ...
}
```

Доступ до адміністування API

З набором властивостей adminAuth Admin API documentation описує як отримати доступ до API.

With the `adminAuth` property set, the Admin API documentation describes how to access the API.

Користувальницька автентифікація

Замість того, щоб користувачам вводити жорстко код в файл налаштувань, можна також підключити спеціальний код для автентифікації користувачів. Це дає змогу інтегруватися з існуючими схемами автентифікації.

Наступний приклад показує, як зовнішній модуль може використовуватися для надання користувацького коду автентифікації.

- Збережіть наступне у файлі з назвою `<node-red>/user-authentication.js`

```
module.exports = {
    type: "credentials",
    users: function(username) {
        return new Promise(function(resolve) {
            // Do whatever work is needed to check username is a valid
            // user.
            if (valid) {
                // Resolve with the user object. It must contain
                // properties 'username' and 'permissions'
                var user = { username: "admin", permissions: "*" };
                resolve(user);
            } else {
                // Resolve with null to indicate this user does not exist
                resolve(null);
            }
        });
    },
    authenticate: function(username,password) {
        return new Promise(function(resolve) {
            // Do whatever work is needed to validate the username/password
            // combination.
            if (valid) {
                // Resolve with the user object. Equivalent to having
                // called users(username);
                var user = { username: "admin", permissions: "*" };
                resolve(user);
            } else {
                // Resolve with null to indicate the username/password pair
                // were not valid.
                resolve(null);
            }
        });
    },
    default: function() {
        return new Promise(function(resolve) {
            // Resolve with the user object for the default user.
            // If no default user exists, resolve with null.
            resolve({anonymous: true, permissions:"read"});
        });
    }
}
```

- Для завантаження цього модулю встановіть властивість `adminAuth` в settings.js:

```
adminAuth: require("./user-authentication")
```

Вузол безпеки HTTP

Маршрути, що виявляються як HTTP In nodes можуть бути захищені за допомогою базової автентифікації.

Для визначення єдиного імені користувача та пароля, яким буде дозволено доступ до маршрутів може бути використана властивість `httpNodeAuth` в файлі `settings.js` .

```
httpNodeAuth:
{user:"user",pass:"$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN."},
```

Властивість pass використовує той самий формат, що й adminAuth. Дивіться Generating the password hash для додаткової інформації.

Доступ до будь-якого статичного вмісту, означеного властивістю httpStatic , може бути забезпечена за допомогою властивості httpStaticAuth, яка використовує той самий формат.

Примітка : у попередніх версіях Node-RED властивість pass очікувалася як хеш MD5. Це криптографічно небезпечно, тому його замінили bcrypt, що використовується adminAuth. Для зворотної сумісності хеш MD5 все ще підтримується, але не рекомендуються до застосування.

Ведення журналу подій (Logging)

За замовчуванням Node-RED використовує реєстратор, який записує його вихід на консоль. Він також підтримує використання користувацьких модулів реєстрації, що дозволяє відправляти вихідні дані в інше місце.

By default, Node-RED uses a logger that writes its output to the console. It also supports the use of custom logger modules to allow the output to be sent elsewhere.

Консольний реєстр (User Guide)

Консольний реєстр може бути налаштований через властивість logging в settings.js.

The console logger can be configured under the logging property in settings.js

```
// Configure the logging output
logging: {
    // Console logging
    console: {
        level: "info",
        metrics: false,
        audit: false
    }
}
```

Для налаштування поведінки журналу використовуються 3 властивості:

There are 3 properties used to configure the logger’s behaviour:

`level`

Рівень реєстрації, який буде записаний. Опції:

- `fatal` - слід записати лише ті помилки, які роблять програму непридатною для використання
- `error` - записати помилки, які вважаються фатальними для конкретного запиту
- `warn` - записувати проблеми, які не є фатальними l
- `info` - запис інформації про загальну роботу програми
- `debug` - записувати інформацію, яка є більш розмовною, ніж інформаційною
- `trace` – дуже делальний запис логів
- `off` – немає логів

Окрім `off`, кожен рівень включає повідомлення на більш високих рівнях - наприклад, `warn` рівень включатиме`error`and `fatal` повідомлення .

Level of logging to be recorded. Options are:

- `fatal` - only those errors which make the application unusable should be recorded
- `error` - record errors which are deemed fatal for a particular request
- `warn` - record problems which are non fatal
- `info` - record information about the general running of the application
- `debug` - record information which is more verbose than info
- `trace` - record very detailed logging
- `off` - no log messages at all

Other than `off`, each level includes messages at higher levels - for example, `warn` level will include `error`and `fatal` level messages.

`metrics`

Коли встановимо `true`, вхідні дані Node-RED дають дані про виконання потоку та використання пам'яті.

Отримані та відправлені події в кожному вузлі виводяться в журнал. Наприклад, з потоку, який має вводити та відлагоджувати вузли, виводяться наступні журнали.

When set to `true`, the Node-RED runtime outputs flow execution and memory usage information.

Received and sent events in each node are output into the log. For example, the following logs are output from the flow which has inject and debug nodes.

```
9 Mar 13:57:53 - [metric]
{"level":99,"nodeid":"8bd04b10.813f58","event":"node.inject.receive","msgid":"86c8
212c.4ef45","timestamp":1489067873391}
9 Mar 13:57:53 - [metric]
{"level":99,"nodeid":"8bd04b10.813f58","event":"node.inject.send","msgid":"86c8212
c.4ef45","timestamp":1489067873392}
9 Mar 13:57:53 - [metric]
{"level":99,"nodeid":"4146d01.5707f3","event":"node.debug.receive","msgid":"86c821
2c.4ef45","timestamp":1489067873393}
```

Використання пам'яті реєструється кожні 15 секунд.

Memory usage is logged every 15 seconds.

```
9 Mar 13:56:24 - [metric]
{"level":99,"event":"runtime.memory.rss","value":97517568,"timestamp":148906778481
5}
9 Mar 13:56:24 - [metric]
{"level":99,"event":"runtime.memory.heapTotal","value":81846272,"timestamp":148906
7784817}
9 Mar 13:56:24 - [metric]
{"level":99,"event":"runtime.memory.heapUsed","value":59267432,"timestamp":1489067
784817}audit
```

Коли встановлено `true`, доступ до Admin HTTP API реєструється. Ця подія включає в себе додаткову інформацію, таку як доступна кінцева точка, IP-адресу та штамп часу.

Якщо `adminAuth` активовано, події містять інформацію про запитуваного користувача.

When set to `true`, the Admin HTTP API access events are logged. The event includes additional information such as the end point being accessed, IP address and time stamp.

If `adminAuth` is enabled, the events include information about the requesting user.

```
9 Mar 13:49:42 - [audit]
{"event":"library.get.all","type":"flow","level":98,"path":"/library/flows","ip":"
127.0.0.1","timestamp":1489067382686}
9 Mar 14:34:22 - [audit]
{"event":"flows.set","type":"full","version":"v2","level":98,"user":{"username":"a
dmin","permissions":"write"},"path":"/flows","ip":"127.0.0.1","timestamp":14890700
62519}
```

Спеціальний модуль реєстрації (User Guide)

Можна також використовувати спеціальний модуль реєстрації. Наприклад metrics вихід може бути відправлений в окрему систему для моніторингу продуктивності системи.

Щоб використовувати користувальницький реєстратор, змініть його settings.js щоб додати новий блок у розділ журналу.

A custom logging module can also be used. For example, the metrics output may get sent to a separate system for monitoring the performance of the system.

To use a custom logger, edit settings.js to add a new block in the logging section.

```
// Configure the logging output
logging: {
    // Console logging
    console: {
        level: "info",
        metrics: false,
        audit: false
    },
    // Custom logger
    myCustomLogger: {
        level: 'debug',
        metrics: true,
        handler: function(settings) {
            return function(msg) {
                console.log(msg.timestamp, msg.event);
            }
        }
    }
}
```

Властивості level, metrics і audit такі ж самі, як і журналу логів консолі.

Властиість handler визначає користувацький обробник журналу. Це функція, яка називається один раз під час запуску, переходячи в конфігурацію реєстратора.

Потрібно повернути функцію, яка буде викликана з повідомленнями журналу.

Можна налаштувати декілька користувацьких реєстраторів - єдине зарезервоване ім'я - console.

The level, metrics and audit properties are the same as console logging.

The handler property defines the custom logging handler. It is a function that is called once at start-up, passing in the logger's configuration. It must return a function that will get called with log messages.

Multiple custom loggers can be configured - the only reserved name is console.

Приклад логера

У наведеному нижче прикладі додано індивідуальний реєстратор, який надсилає події метрики в екземпляр logstash через TCP-з'єднання.

Це дуже швидкий і простий приклад - без обробки помилок або повторної підключення логіки.

The following example adds a custom logger that sends metrics events to a logstash instance over a TCP connection.

It is a very quick and simple example - with no error handling or reconnect logic.

```
logging: {
    console: {
        level: "info",
        metrics: false,
        audit: false
    },
    logstash: {
        level:'off',
        metrics:true,
        handler: function(conf) {
            var net = require('net');
            var logHost = '192.168.99.100',logPort = 9563;
            var conn = new net.Socket();
            conn.connect(logPort,logHost)
                .on('connect',function() {
                    console.log("Logger connected")
                })
                .on('error', function(err) {
                    // Should attempt to reconnect in a real env
                    // This example just exits...
                    process.exit(1);
                });
            // Return the function that will do the actual logging
            return function(msg) {
                var message = {
                    '@tags': ['node-red', 'test'],
                    '@fields': msg,
                    '@timestamp': (new Date(msg.timestamp)).toISOString()
                }
                try {
                    conn.write(JSON.stringify(message)+"\n");
                }catch(err) { console.log(err);}
            }
        }
```

```
    }
}
```

Адміністратор командного рядка (Command-line Administration)

Інструмент командного рядка node-red-admin дозволяє віддалено керувати екземпляром Node-RED.

The node-red-admin command-line tool allows you to remotely administer a Node-RED instance.

Усановка (User Guide)

Встановіть це глобально, щоб зробити node-red-admin Команда доступна на вашому шляху:

Install this globally to make the node-red-admin command available on your path:

```
npm install -g node-red-admin
```

Примітка : sudo потрібен, якщо він працює як некористувацький користувач на Linux/OS X. Якщо він працює на Windows, вам доведеться запускати в command shell as Administrator, без команди sudo.
Note : sudo is required if running as a non-root user on Linux/OS X. If running on Windows, you will need to run in a command shell as Administrator, without the sudo command.

Об’єкт та вхід (User Guide)

Щоб дистанційно керувати екземпляром Node-RED, інструмент має вказуватись на екземпляр Node-RED, до якого ви хочете отримати доступ. За замовчуванням він передбачаєhttp://localhost:1880. Щоб змінити це, використовуйт команду target :

To remotely administer a Node-RED instance, the tool must first be pointed at the Node-RED instance you want it to access. By default, it assumes http://localhost:1880. To change that, use the target command:

```
node-red-admin target http://node-red.example.com/admin
```

Якщо автентифікація увімкнено, ви повиннівести login:

If authentication is enabled, you must then login:

```
node-red-admin login
```

Ці команди створюють файл з назвою ~/.node-red/.cli-config.json що зберігає об’єкт та отримує інформацію про токен.

These commands create a file called ~/.node-red/.cli-config.json that stores the target and access token information.

Примітка : Параметр `hash-pw` *не* вимагає, щоб інструменту для входу в систему і може бути запущений у будь-який час.

Note : The `hash-pw` option does *not* require the tool to be logged in and can be run at any time.

Інші команди (User Guide)

Інструмент надає наступні команди:

- list - Перелік усіх встановлених вузлів
- info - Показати додаткову інформацію про модуль або набір вузлів
- enable - Увімкнути вказаний модуль або набір вузлів
- disable - Вимкнути вказаний модуль або вузол
- search - Пошук NPM для модулів Node-RED, що стосуються даного пошукового терміна
- install - Встановити модуль з NPM
- remove - Видалити модуль NPM
- hash-pw - Створіть хеш пароля, який можна використовувати з adminAuth і httpNodeAuth налаштуванням

The tool provides the following commands:

- list - List all of the installed nodes
- info - Display more information about the module or node set
- enable - Enable the specified module or node set
- disable - Disable the specified module or node set
- search - Search NPM for Node-RED modules relating to the search-term given
- install - Install a module from NPM
- remove - Remove an NPM module
- hash-pw - Create a password hash that can be used with the adminAuth and httpNodeAuth settings

Розширення (Advanced)

Цей розділ є необробленим.

Вставка до існуючого додатку (Embedding into an existing app)

Можна вставляти Node-RED у більшу програму. Типовим сценарієм буде те, що ви використовуєте Node-RED для створення потоків даних, які ви хочете відображати на веб-панелі, - все з тієї ж програми.

It is possible to embed Node-RED into a larger application. A typical scenario would be where you use Node-RED to generate flows of data that you want to display on a web dashboard - all from the same application

Додати `node-red` до залежностей модуля у вашій програмі `package.json`, разом з будь-якою з окремих залежностей вузла ви можете мати.

Add `node-red` to the module dependencies in your application's `package.json`, along with any of the individual node dependencies you may have.

Нижче наводиться мінімальний приклад вбудованого середовища виконання у ширшу програму Express.

The following is a minimal example of embedded the runtime into a wider Express application.

```
var http = require('http');
var express = require("express");
var RED = require("node-red");

// Create an Express app
var app = express();

// Add a simple route for static content served from 'public'
app.use("/",express.static("public"));

// Create a server
var server = http.createServer(app);

// Create the settings object - see default settings.js file for other options
var settings = {
    httpAdminRoot:"/red",
    httpNodeRoot: "/api",
    userDir:"/home/nol/.nodered/",
    functionGlobalContext: { }    // enables global context
};

// Initialise the runtime with a server and settings
RED.init(server,settings);

// Serve the editor UI from /red
app.use(settings.httpAdminRoot,RED.httpAdmin);

// Serve the http nodes UI from /api
app.use(settings.httpNodeRoot,RED.httpNode);

server.listen(8000);

// Start the runtime
RED.start();
```

Коли цей підхід використовується, файл `settings.js` включаючи і Node-RED не використовується. Замість цього налаштування передаються викликом `RED.init` як показано вище.

When this approach is used, the `settings.js` file included with Node-RED is not used. Instead, the settings are passed to the `RED.init` call as shown above.

Крім того, наступні параметри ігноруються, оскільки вони залишаються вам для налаштування екземпляра Express, як вам це потрібно:

- `uiHost`

- `uiPort`
- `httpAdminAuth`
- `httpNodeAuth`
- `httpStatic`
- `httpStaticAuth`
- `https`

Furthermore, the following settings are ignored as they are left to you to configure the Express instance as you want it:

- `uiHost`
- `uiPort`
- `httpAdminAuth`
- `httpNodeAuth`
- `httpStatic`
- `httpStaticAuth`
- `https`

Dashboard (node-red-dashboard)

Модуль **dashboard** (**приладова панель**) надає набір вузлів в Node-RED для швидкого створення інформаційної панелі.
З версії 2.10.0 можна створювати та встановлювати вузли віджетів, подібно до інших Node-RED вузлів. Дивіться Wiki для отримання додаткової інформації.

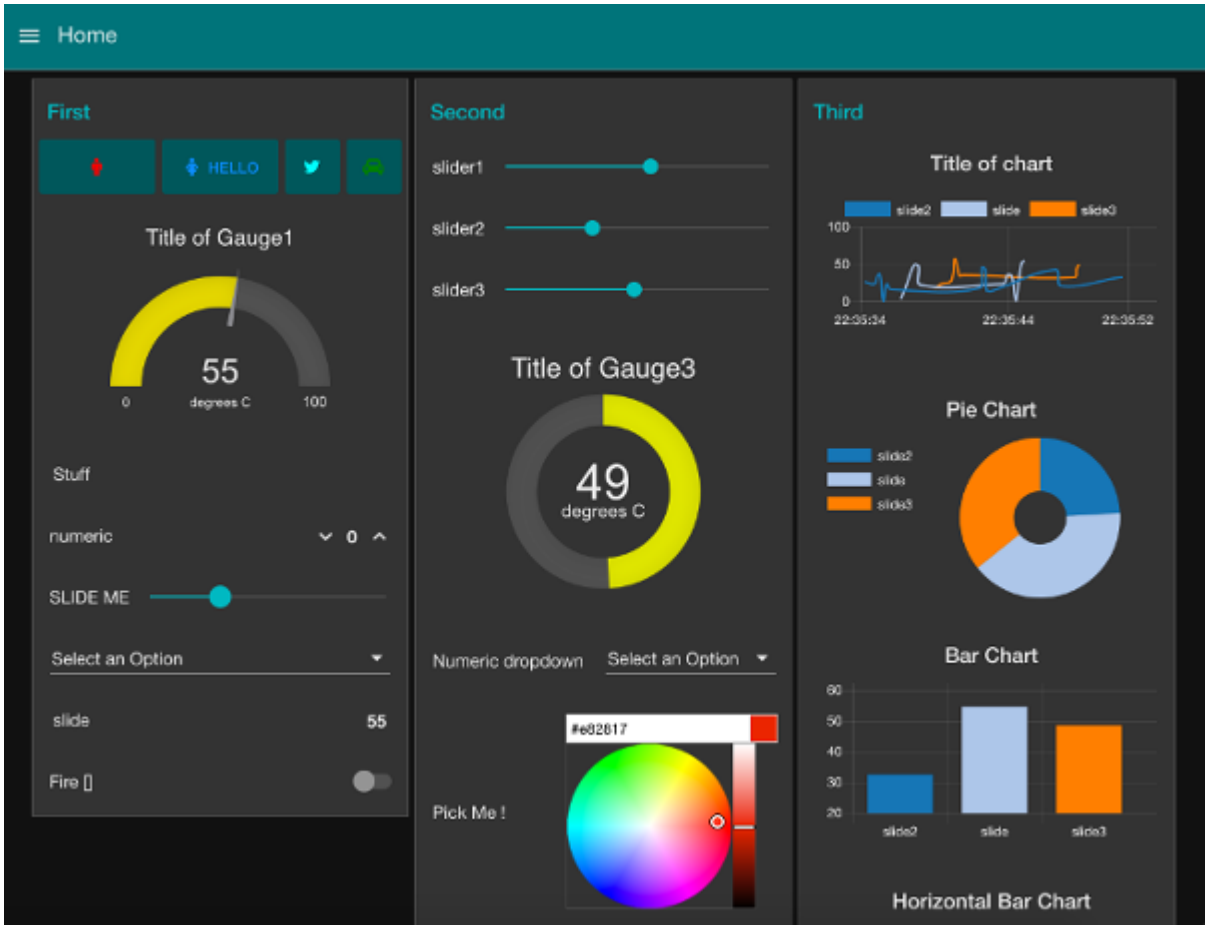


рис.1. Приклад приладової панелі

Додаткові посилання:
https://flows.nodered.org/node/node-red-dashboard
https://github.com/node-red/node-red-dashboard/wiki/Creating-New-Dashboard-Widgets

Інсталяція

Щоб встановити стабільну версію модуля, скористайтеся параметром `Menu - Manage palette` та пошуком версії для `node-red-dashboard` або запустіть наступну команду у каталозі користувача Node-RED (зазвичай ~ / .node-red):

```
npm i node-red-dashboard
```

Перезапустіть екземпляр Node-RED, і у палітрі повинні бути доступні вузли користувальницького інтерфейсу UI та на правій бічній панелі нова вкладка `dashboard`.

Інтерфейс UI доступний за адресою http://localhost:1880/ui (якщо використовуються параметри за замовчуванням).

Якщо ви хочете спробувати нову версію з github, її можна встановити за допомогою

```
npm i node-red/node-red-dashboard
```

Компонування та налаштування

Типова URL-адреса для інформаційної панелі базується на наявному Node-RED httpRoot-шляху, додавши /ui. Це можна змінити у файлі settings.js Node-RED - `ui:{path:"ui"}`.
Основні властивості, що відносяться до Dashboard налаштовуються через відповідний пункт меню бічної панелі.

Компонування приладової панелі розглядається у вигляді сітки, до якої прив'язуються елементи (віджети). Сітка вимірюється в одиницях сітки (**unit)**, які за замовченням рівні 48px шириною з 6px зазором. За необхідності параметри одиниці та зазори можна змінити в налаштуваннях dashboard -> Site

рис.2. Налаштування одиниці стіки.

Приладовий інтерфейс ділиться на вкладки (окремі сторінки), які створюються і налаштовуються на вкладці бічної панелі Layout (компонування). В налаштуваннях можна змінити порядок вкладок, груп і віджетів, і добавити та редагувати інші властивості. Також тут можна добавити в меню **посилання** (**Links**) на інші веб-сторінки. Це може опціонально бути відкритим в iframe - якщо дозволено цільовою сторінкою.

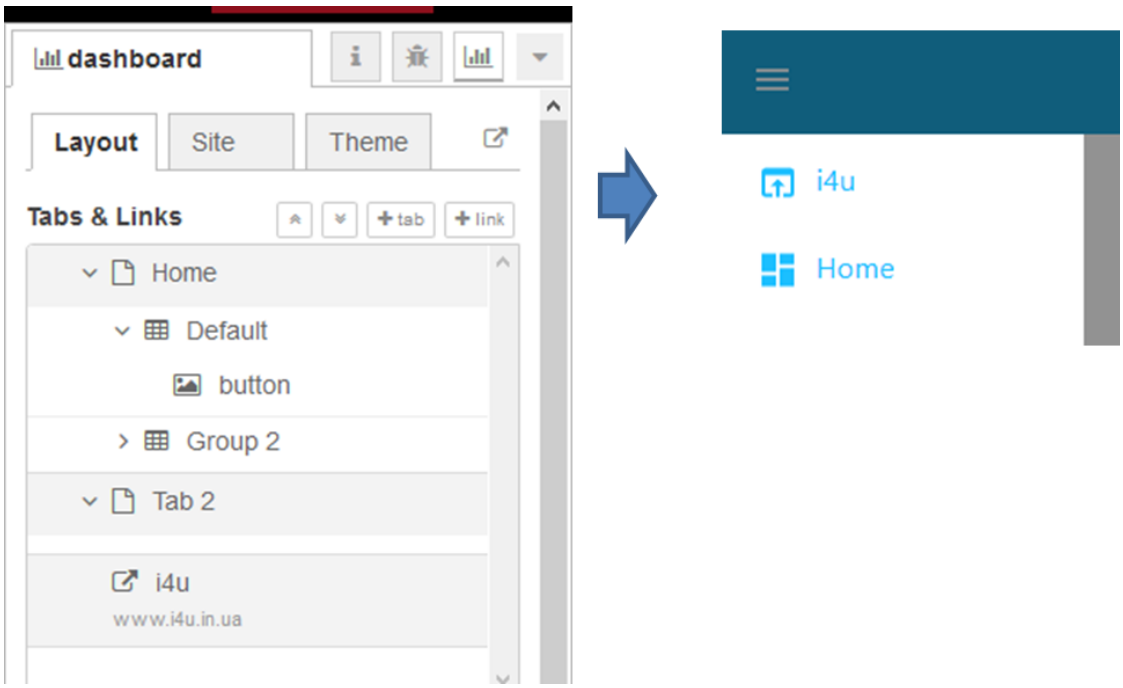


рис.3. Вкладки.

В налаштуваннях Site доступні для зміни також інші властивості, зокрема заголовок (Title), опції (Options) та формат дати (Date Format).

.

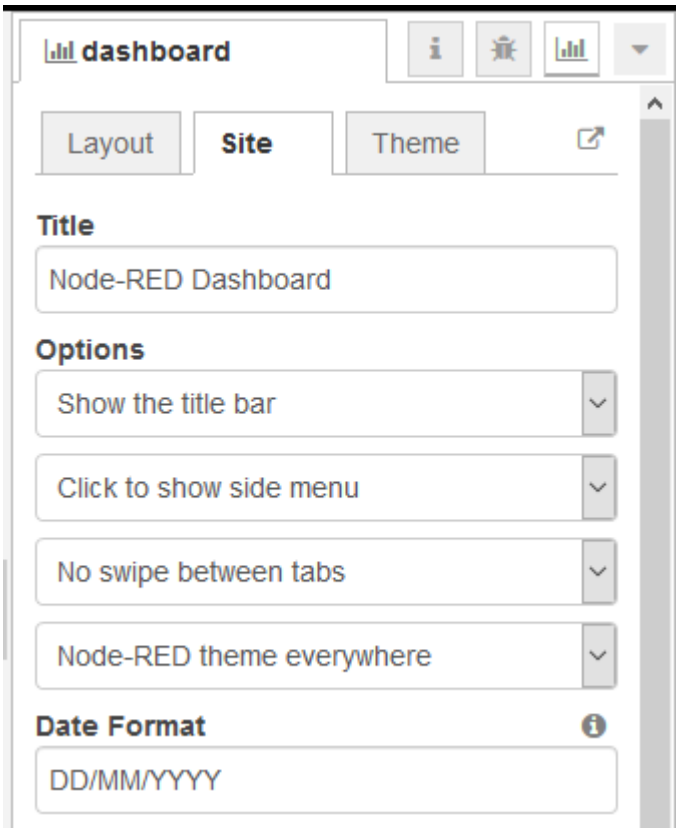


рис.4. Властивості сайту.

Опції дозволяють сховати рядок заголовка, дозволити переміщуватися збоку між вкладками на сенсорному екрані. Можна також встановити та налаштувати тему, що використовується для відображення.

Елементи (**віджети**) об’єднуються в **групи** (**group**), для яких вказується ширина в одинцях сітки. За замовченням це 6 одиниць.

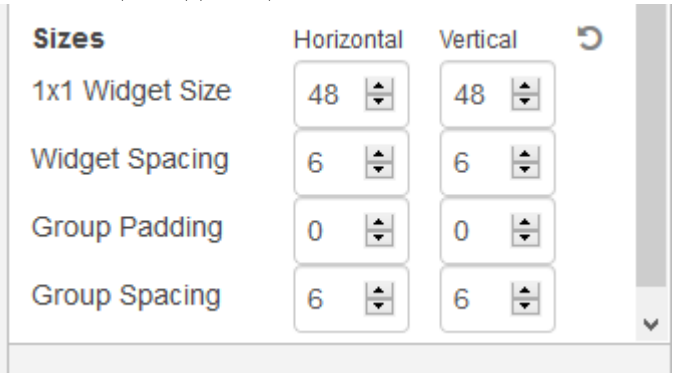


рис.5. Налаштування розміщення груп одна відносно одної в налаштуваннях dashboard -> Site.

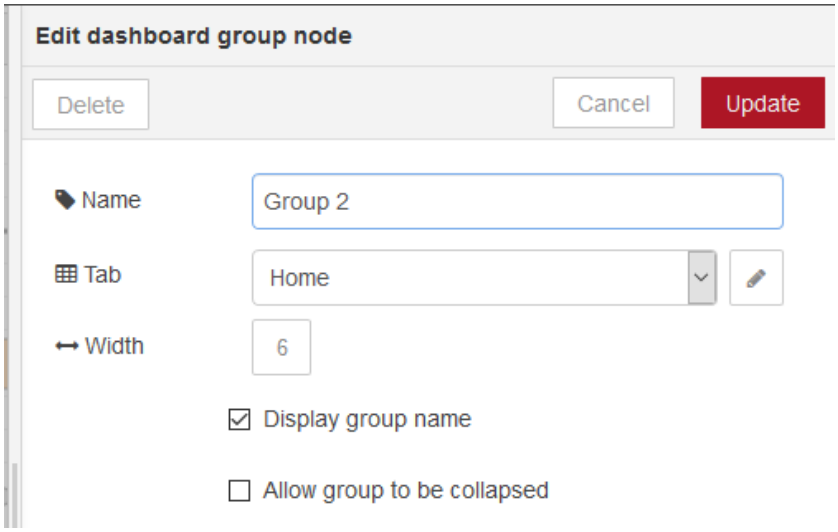


рис.6. Налаштування розміру групи

Кожен елемент на панелі, що представлений вузлом, - **віджет** (**widget**), розміщується в конкретній групі, яка вибирається в його налаштуваннях (рис.7). Для віджета налаштовується ширина, яка за замовченням вказана як «авто» - це значить що елемент буде заповнюватися по ширині групи, в якій знаходиться. Значення ширини можна також встановити її фіксованою в кількістях одиниць.

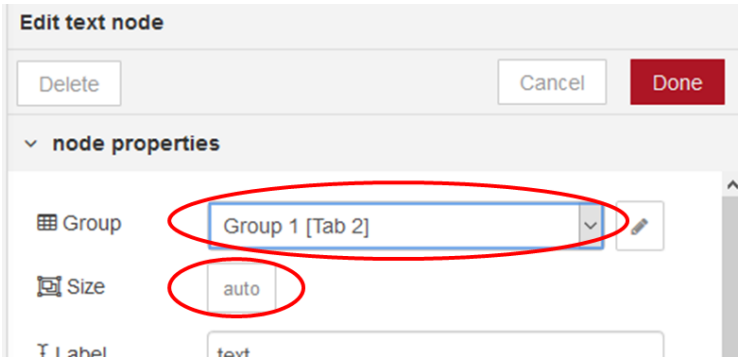


рис.7. Налаштування розміщення віджета в групі

Алгоритм компонування приладової панелі завжди намагається помістити елементи в верхньому лівому кутку контейнеру - це відноситься як до розміщення груп на сторінці, так саме і до розміщення віджетів в групі. Тобто, з урахуванням групи шириною 6, якщо Ви добавляєте шість віджетів, кожен з шириною 2, тоді вони будуть викладені в два рядки – по три віджета в кожному. Якщо Ви добавляєте дві групи з шириною 6, вони будуть розміщуватися один біля одного до тих пір, поки дозволяє ширина вікна браузера. Якщо Ви зменшуєте ширину вікна браузера, в деякий момент друга група буде зміщена нижче першої в одну колонку. Це дозволяє використовувати декілька груп якщо це можливо, а не одну велику групу, так щоб сторінки могли динамічного змінювати розмір для малих екранів.

На вкладці Theme можна означити стилі сторінок.

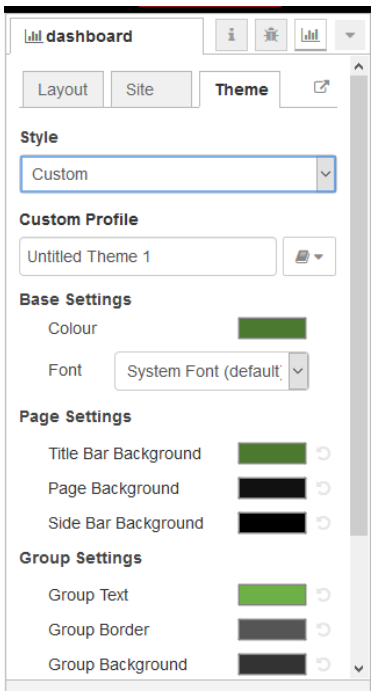


рис.8. Налаштування стилю

Налаштування віджетів (Widgets)

Кожен віджет крім налаштувань розміщення в групі, які описані вище, має ряд властивостей, що означують його поведінку. Більшість віджетів можуть мати мітку (label) та значення (value) - обидва, якщо це потрібно, можуть бути означені як статичним текстом, так і властивістю вхідного повідомлення (msg), і модифіковані ангулярними виразами (детально про використання ангулярних виразів читай в наступному пункті).

Наприклад, мітку можна встановити властивістю повідомлення `topic` а значення в `payload`, це робиться через означення їх у подвійних фігурних дужках, тобто `{{msg.topic}}` та `{{msg.payload}}`, як це показано на рисунку.

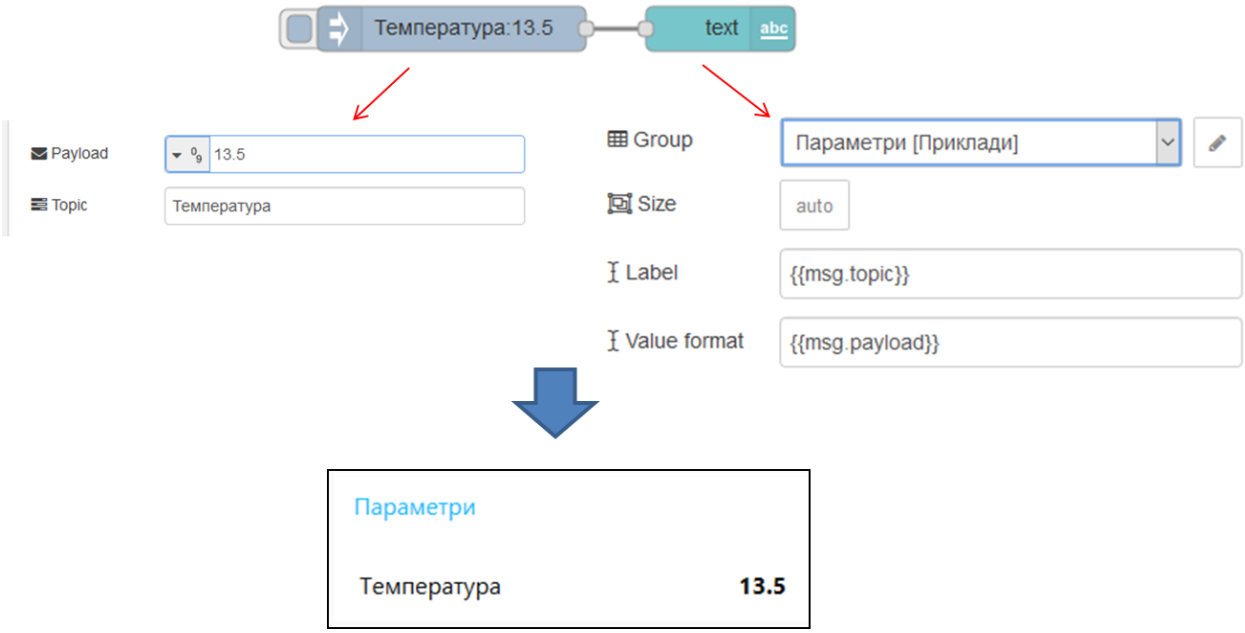


рис.9. Налаштування властивостей віджета через властивості вхідного повідомлення.

Будь-який віджет може бути деактивований шляхом передачі властивості `msg.enabled = false`. Це не зупиняє отримання віджетом повідомлень але зупиняє активність вводу і змінює стиль віджета.

Деякі віджети, що мають графічне зображення (наприклад кнопка) мають налаштування піктограми (**Icon**), яку можна означити через назву. У версії Dashboard 2.11 піктограми доступні з сайтів Material Design icon *(наприклад 'check', 'close')* або Font Awesome icon *(наприклад 'fa-fire')* або Weather icon. Наприклад, якщо зайти на сайт Material Design icon можна знайти піктограму з назвою «trending_up». Якщо цю назву вписати у властивість кнопки Icon, біля тексту з'явиться вибране зображення (див. рис.10).

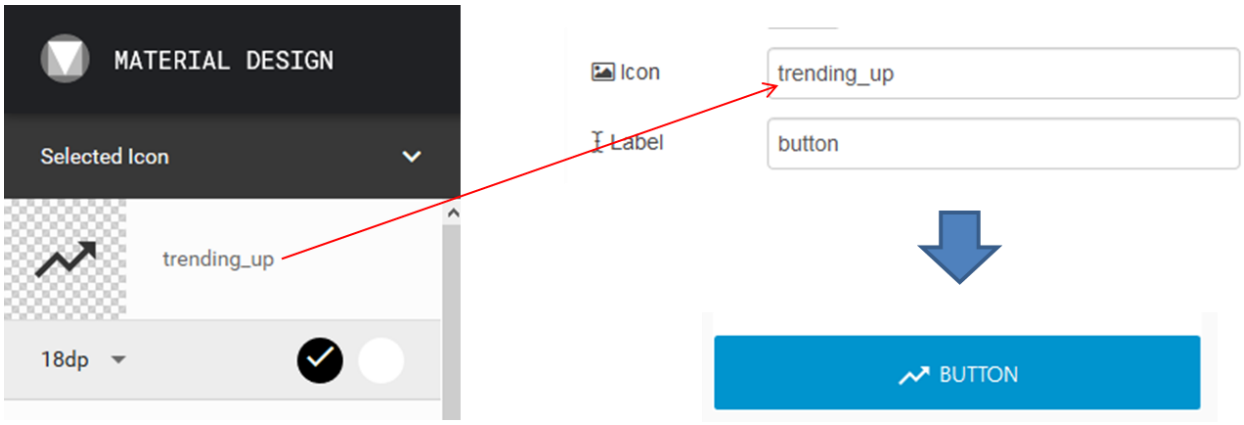


рис.10. Приклад використання піктограм в зображенні кнопки.

Основи AngularJS в Node-RED dashboard

Про AngularJS

Модуль Node-RED Dashboard базується на шаблонах AngularJS. AngularJS – це фреймворк для скриптів JavaScript, що вбудовуються в HTML і виконуються на стороні WEB-клієнта. AngularJS базується на концепції MVC (Модель-вигляд-контролер), в якій застосунок умовно ділиться на три взаємопов'язані частини: модель даних; вигляд (зовнішній інтерфейс); модуль керування (контролер), що здійснює обробку подій від інтерфейсу користувача та зв'язує вигляд з даними. У AngularJS зовнішній вигляд сторінки є проекцією моделі даних через шаблон HTML. Це означає, що кожного разу, коли дані змінюються, AngularJS оновлює відповідні точки зв'язування, які змінюють вигляд. Точки зв'язування можуть проводитись через прив'язування даних (data binding), канали (pipes) та директиви (directives).

Користуватися Dashboard в Node-RED можна і без розуміння AngularJS. Нижче пояснюються деякі основи синтаксису прив'язки даних та використання фільтрів, що потребується для налаштування зв'язування віджетів з даними та їх відображення. Більш глибокі знання AngularJS потрібні для створення власних віджетів та використання шаблонів.

Прив'язування даних (Data binding)

AngularJS підтримує прив'язування даних (як одностороннє так і двостороннє), механізм координації частин шаблону з частинами компонента. До шаблону HTML додається розмітка прив'язування (спеціальні символи), щоб сказати Angular де і як підключити дані. Наступна діаграма показує чотири форми розмітки прив'язування даних. Кожна форма має напрямок: до DOM, від DOM або до обох.

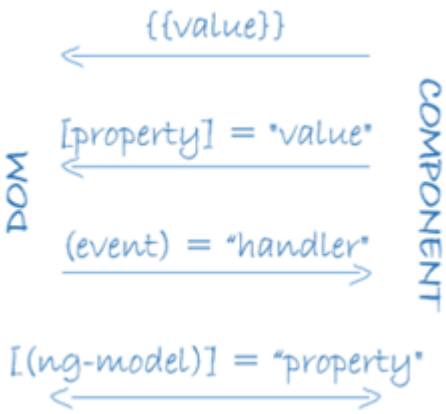


рис.11. Форми прив'язування даних та використовувана розмітка.

Один з видів прив'язування є підстановочні вирази (Interpolation), які знаходяться в потрібному місці HTML і виділяються подвійними фігурними дужками "{{value}}". Таким чином, все що знаходиться в цих дужках буде вираховуватися як Angular вираз, а результат розміщуватися у вказаному місці. У Node-RED, для прикладу Label (мітка) в налаштуваннях віджета може бути встановлена в

{{msg.topic}}

, що буде значити, що значення властивості вхідного повідомлення msg.topic буде використовуватися у відображенні мітки.

Кожен вузол віджета може парсити `msg.payload` щоб зробити його розміщення на дисплеї. Ця перетворена версія представляється як змінна що називається `value`. Таким чином в замість {{`msg.payload`}} можна використовувати підстановку {{`value`}}.

Інші форми прив'язування описані в розділі Template.

Angular вирази – це вирази, написані на JavaScript, що можуть використовуватися в прив'язуваннях. Однак далеко не всі вирази JavaScript можуть використовуватися в якості підстановочних. Зокрема можна використовувати арифметичні операції, виклик деяких функцій. Не дозволяється використовувати:

- присвоєння (=, +=, -=, ...)
- `new`
- послідовність виразів з символами «;» або «,»
- операції increment та decrement (++ та --)
- побітові оператори «|» та «&»
- нові оператори такі як «|», «?» «.» та «!».

Ангулярні фільтри

У Node-RED, вирази можуть включати фільтри, які форматують (перетворюють) значення виразу для відображення його користувачеві. Це реалізовано через механізм AngularJS каналів (pipe). У Node-RED dashboard фільтри можуть бути використані в шаблонах представлень. Загальний запис застосування фільтра в виразах на сторінці представлення виглядає так:

{{ expression | filter }}

де «|» - це оператор каналу, filter – застосований фільтр

Наприклад, якщо записати {{12 | currency}}, то буде застосований вбудований фільтр для відображення валюти. У результаті ми побачимо (в залежності від локальних налаштувань) - $ 12.00.

Фільтрів може бути багато:

{{ expression | filter1 | filter2 | ... }}

У цьому випадку перший результат буде оброблений другим фільтром і т.д .. Такий тип запису і обробки називається 'chaining'.

Ще у фільтрів можуть бути аргументи.

{{ expression | filter: argument1: argument2:... }}

Наприклад для відображення числового значення може бути використаний фільтр `number, який` дозволяє форматувати числа. Наприклад,

```
{{234.5677 | number:1}}%
```

Результатом роботи даного фільтра буде число 234,6 доповнено знаком %. Числовий параметр після двокрапки вказує, скільки залишиться чисел після коми (умовною, оскільки в даному випадку для поділу цілої і дробової частини використовується точка). Якщо числовий параметр більше кількості розрядів після коми, то при виведенні число доповнюється нулями. Для введення спеціальних символів можна використовувати їх назву або код HTML, наприклад: &deg покаже символ градуса/

Використовуючи фільтри `lowercase` і `uppercase`, ми можемо приводити вміст до нижнього і верхнього регістру відповідно. Наприклад,

```
{{question.text | lowercase}}
```

Щоб вивести дату в певному форматі застосовується фільтр `date`. Як вираз використовується кількість мілісекунд, що пройшли з початку епохи Unix (тобто з 1 січня 1970 року). Наприклад, якщо в вузлі Text означити Value як вихід з вузла Inject типу Timestamp, використовуючи фільтр {{value | date:'dd.MM.yyyy hh:mm:ss'}} , то ми отримаємо значення дати та часу.

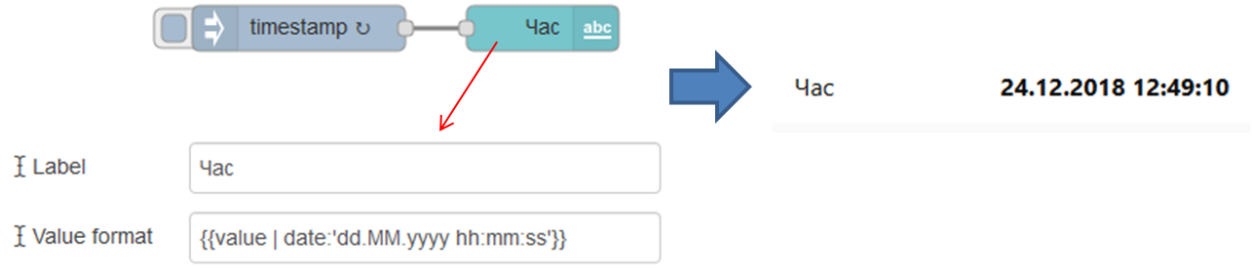
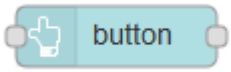


рис.12. Приклад використання фільтру для виведення відмітки часу у форматі дати та часу.

Повний перелік AngularJS каналів даний за цим посиланням.

Button (Кнопка)



Додає до інтерфейсу користувача кнопку.

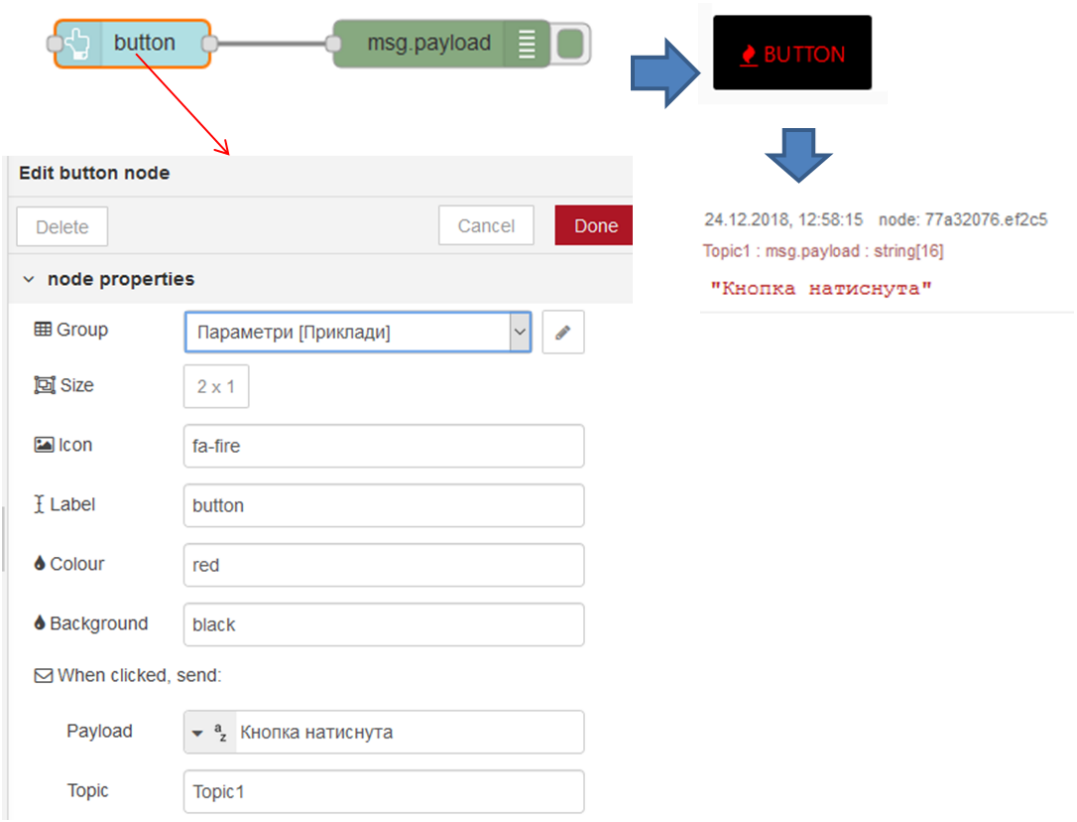


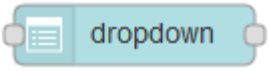
рис.13. Налаштування та зовнішній вигляд кнопки.

Натискання кнопки генерує повідомлення з встановленим `msg.payload` рівним значенню в полі Payload. Якщо в налаштуваннях це поле не вказано, використовується ідентифікатор вузла (node id).

Можна встановити кольори тексту та фону введенням їх символьної назви або числового представлення в HTML. Їх також можна змінювати властивістю повідомлення, встановивши в полі значення ім'я властивості, наприклад `{{msg.background}}`.

Якщо встановлено опцію «If `msg` arrives on input, pass through to output:» то повідомлення на вході буде ініціювати дію, аналогічну натисканню кнопки. Вихід payload при цьому буде таким, як означено в конфігурації вузла.

Dropdown (Спадне меню)



Додає до інтерфейсу користувача спадне меню вибору.

За потребою можуть бути додані кілька пар значення/мітки. Якщо мітка не вказана, буде використано значення в якості відображення мітки.

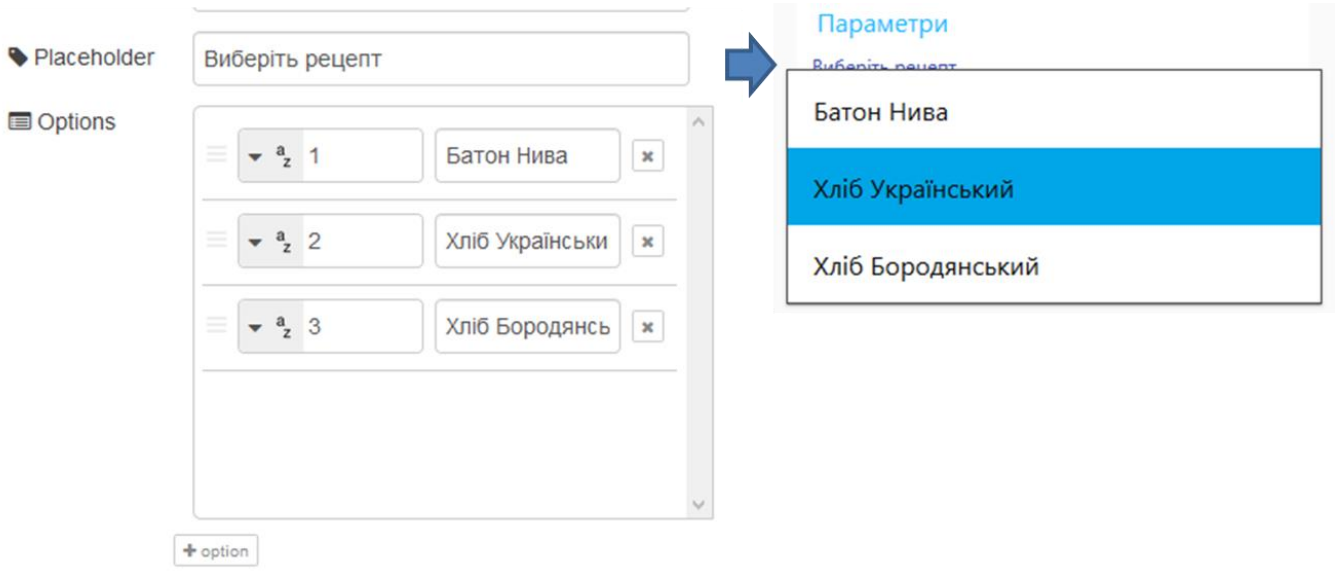


рис.14. Налаштування та зовнішній вигляд спадного меню.

Налаштоване значення вибраного елемента буде повернуто як `msg.payload`. Встановлення на вході `msg.payload` значення переведе спадне меню у це значення.

Параметри (Options) можуть бути також налаштовані шляхом подачі на вузол повідомлення `msg.options`, що містять масив. Якщо подається просто текст, то значення буде таким же, як і мітка. Для встановлення і мітки значення необхідно використовувати пару "label": "value". Наприклад, якщо налаштувати вузли як показано на рис.15, то при ініціюванні Inject, в спадаючий список буде надіслано вказаний в `msg.options` вузлом Change список рецептів, та вибраний 2-й із них, так як `msg.payload=«2»`

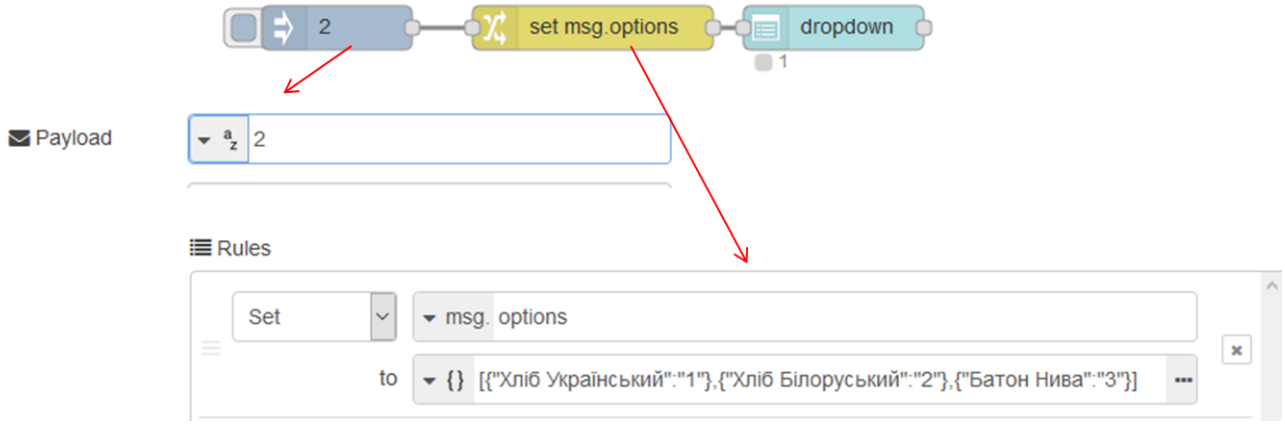
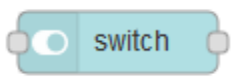


рис.15. Приклад використання msg.options.

Switch (Перемикач)



Додає до інтерфейсу користувача перемикач.

Кожна зміна стану перемикача буде генерувати `msg.payload` з вказаними значеннями On і Off.

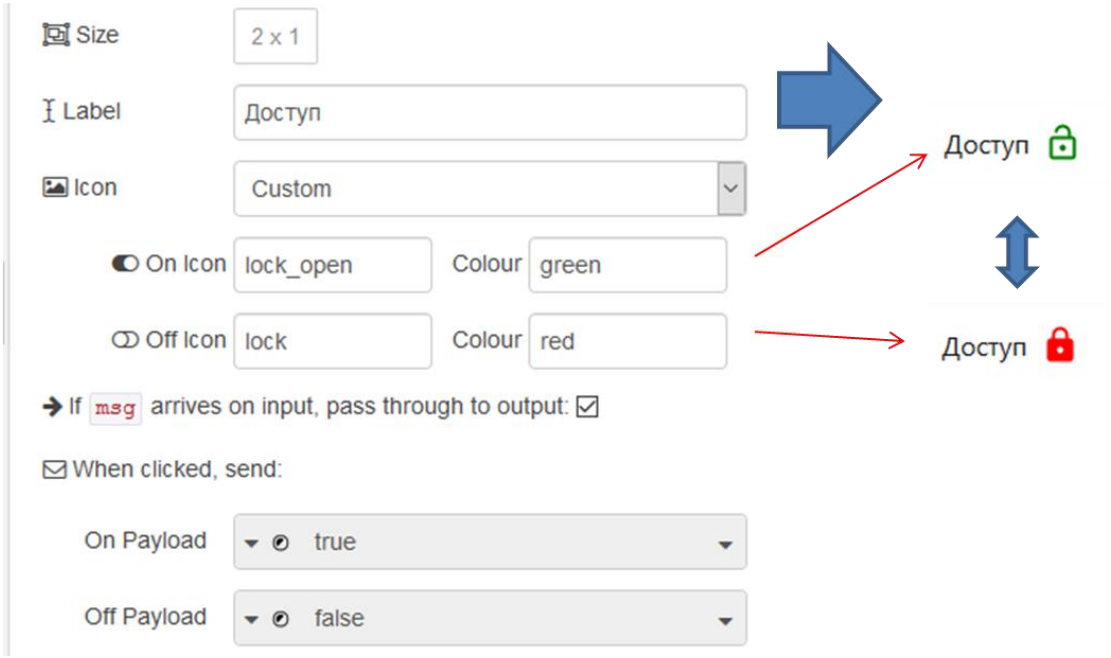


рис.16. Налаштування Switch.

Колір увімкнення/вимкнення та значок увімкнення/вимкнення є необов'язковими. Якщо вони є присутніми, перемикач за замовчуванням буде замінено відповідними іконками та відповідними кольорами (див. рис.16).

Slider (Повзунок)

Додає до інтерфейсу користувача віджет повзунка.

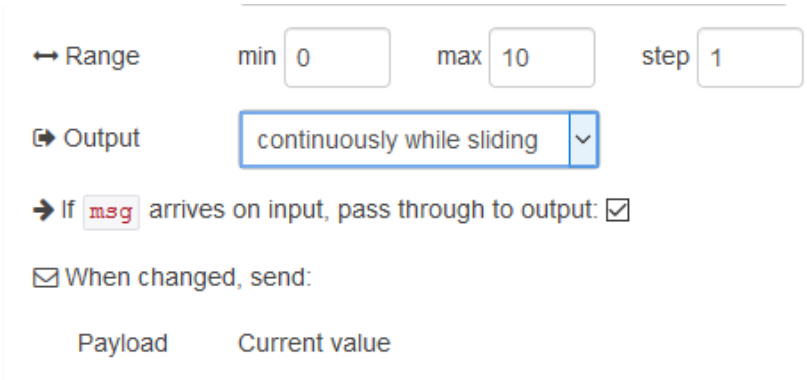


рис.17. Налаштування Slider.

Користувач може змінювати своє значення в межах що встановлені налаштуванням min та max з кроком step. Кожна зміна значення буде генерувати повідомлення зі значенням, встановленим як payload.

Якщо перетворення не вдасться, буде використано значення min. Якщо значення змінюється, воно також передається на вивід.

Примітка: вхідне повідомлення msg для вузла повзунка не змінить відображення інформації по стан, якщо вихід вузла повзунка не з'єднаний з іншим вузлом.

Numeric (Числове поле)

Додає до інтерфейсу користувача віджет зміни числового значення кнопками «більше» та «менше». Користувач може встановити значення в межах min та max та кроком step. Кожна зміна значення буде генерувати `msg.payload`.

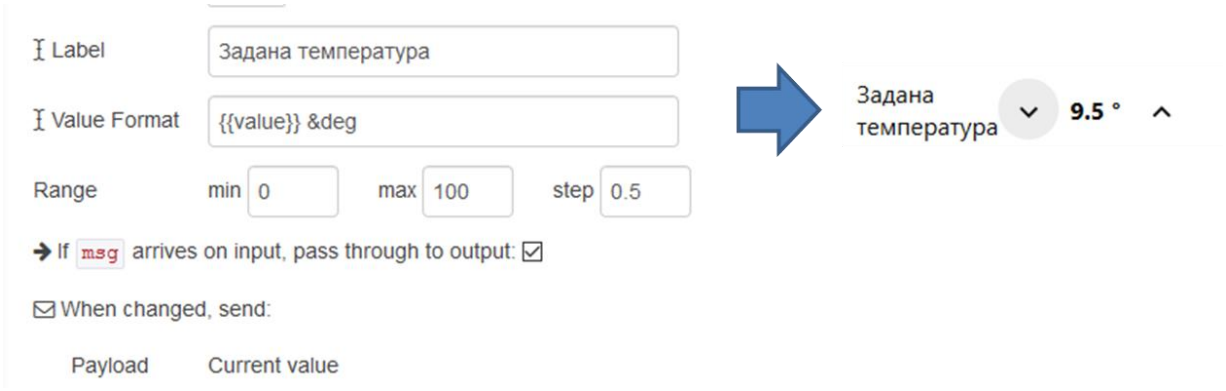
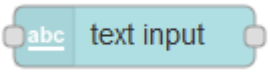


рис.18. Налаштування Numeric.

Будь-які вхідні повідомлення будуть перетворені в число. Якщо перетворення не вдасться буде використано значення min, і оновить інтерфейс користувача. Якщо значення змінюється, воно також передається на вивід.

Text input (Текстове поле для вводу)



Додає до інтерфейсу користувача поле введення тексту, електронної пошти або вибору кольорів.

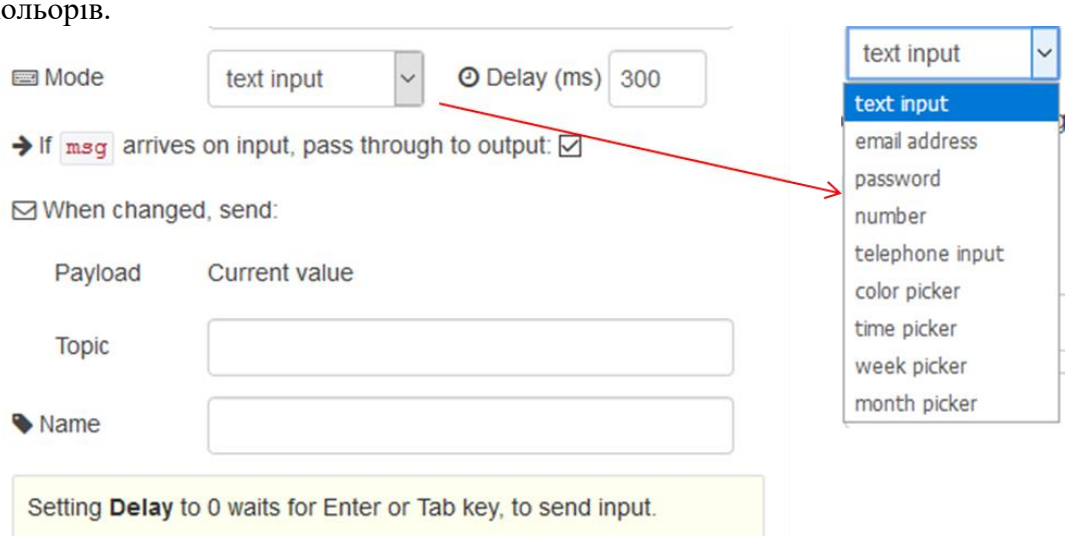
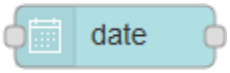


рис.18. Налаштування Text input.

Будь-яке вхідне повідомлення автоматично надсилається як `msg.payload`. Ви також можете задати текст поля введення, надіславши повідомлення `msg.payload`. Затримка **Delay** (за замовчуванням: 300ms) встановлює час у мілісекундах перед надсиланням на вихід після введення останнього символу. Встановлення значення 0 очікує клавіші "Enter" або "Tab" щоб зробити надсилання.

Властивість Mode задає режим роботи текстового ввода, який по суті призначений для означення формату. Так, наприклад, при виборі режиму електронної пошти, поле введення буде забарвлено в червоний колір, якщо формат введеної користувачем пошти буде невірним, і повернеться на вихід вузла як «невизначено». Не всі браузери підтримують типи вводу тижня та місяця і можуть повертати як невизначені.

Date picker(Вибір дати)



Додає до інтерфейсу користувача віджет вибору дати. Відображення дати можна відформатувати на вкладці "Панель інструментів - сайт" за допомогою форматування moment.js. Наприклад, `MM/DD/YYYY`, `Do MMM YYYY` or `YYYY-MM-DD`.

Colour picker (Вибір кольору)

Додає до інтерфейсу користувача панель вибору кольору.

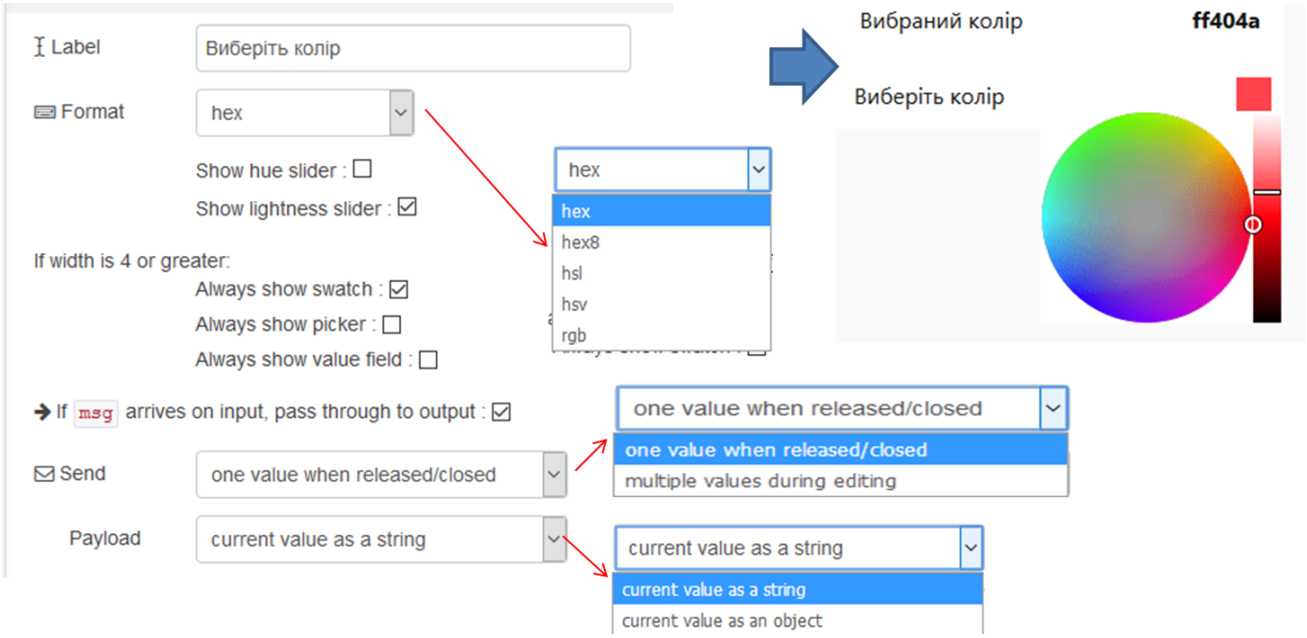


рис.19. Налаштування та вигляд Colour picker.

Якщо ширина групи становить 4 або більше, то панель вибору кольору буде завжди видимою. Формат може бути rgb, hex, hex8, hsv або hsl. Прозорість підтримується для всіх, крім hex.

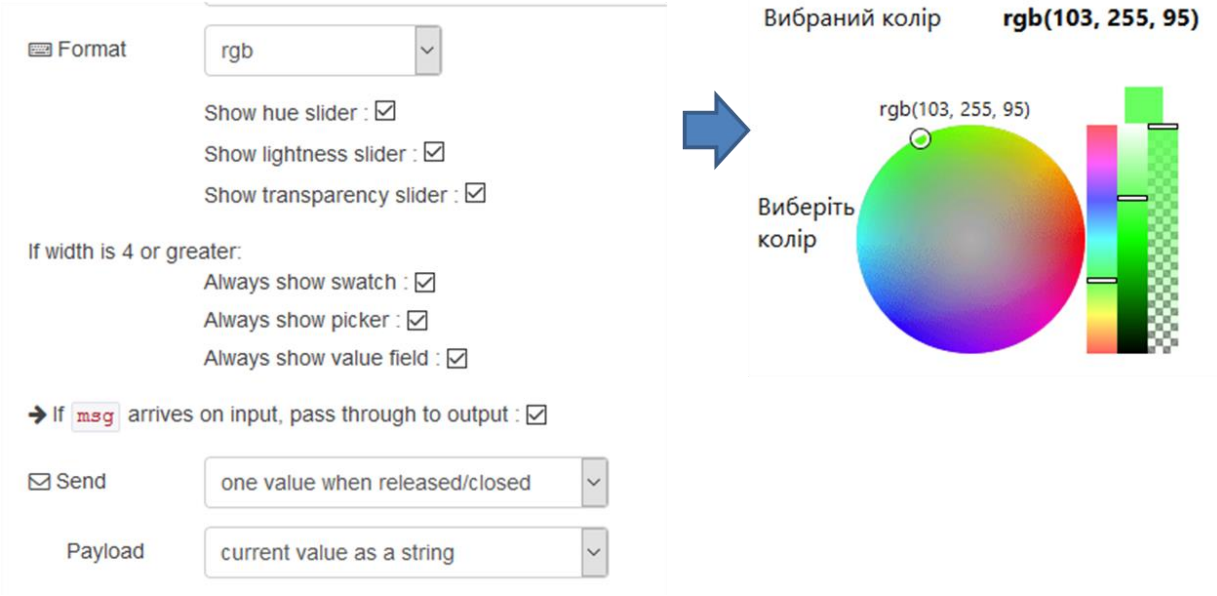


рис.20. Налаштування та вигляд Colour picker з усіма видимими повзунками налаштування.

Form (Форма)

Додає до інтерфейсу користувача форму.

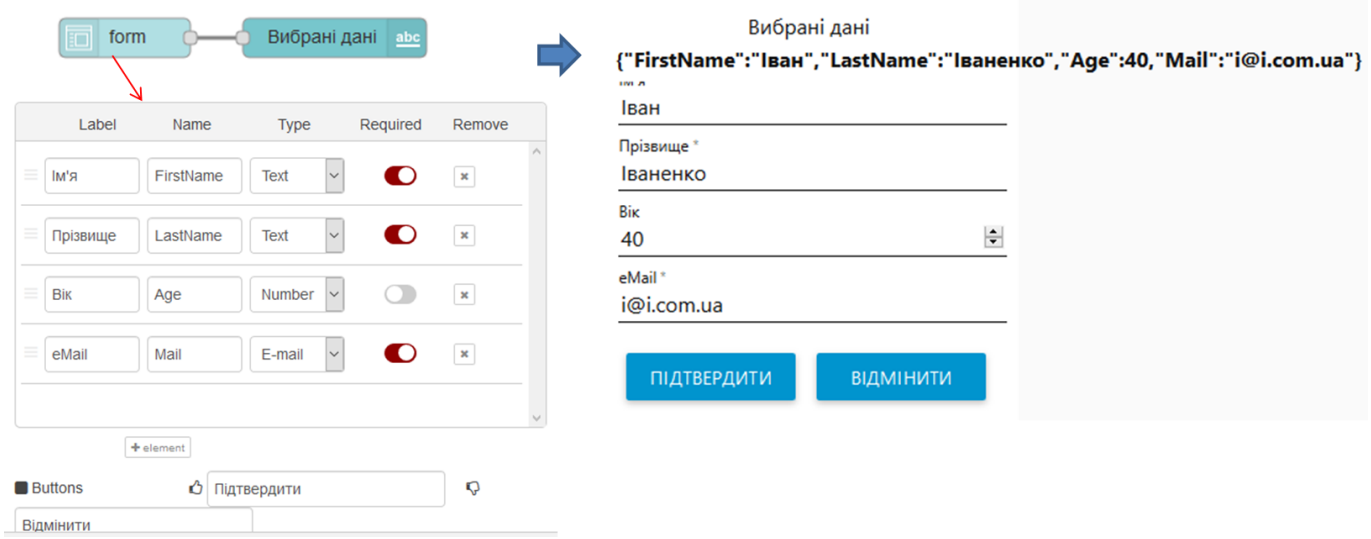


рис.21. Налаштування та вигляд Form.

Допомагає зібрати різні значення від користувача і при натисканні кнопки «submit» завантажує їх в `msg.payload` у вигляді JSON. Необхідні елементи добавляються на форму використовуючи кнопку «+element».

Кожен елемент вміщує наступні компоненти:

- Label : Значення, що буде відображатися користувачу в полі для введення
- Name : Ім’я змінної (ключ) що буде властивістю `msg.payload` в якій буде знаходитися введене значення
- Type : Тип поля для введення
- Required : Чи обов’язкове поле
- Delete : Видалити елемент

Text (Вивід тексту)

Додає до інтерфейсу користувача поле для виведення тексту.
Кожний отриманий `msg.payload` оновить текст на основі наданого формату значення **Value Format**.

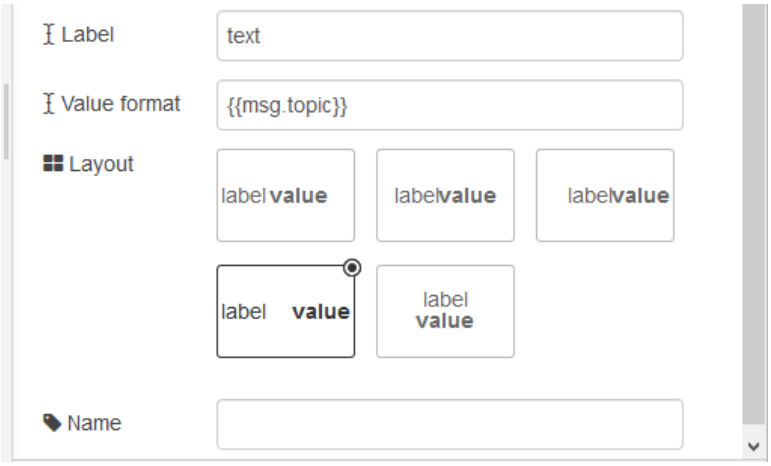
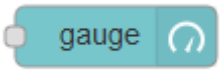


рис.22. Налаштування Text.

Gauge (Індикатор)



Додає до інтерфейсу користувача віджет індикатору.

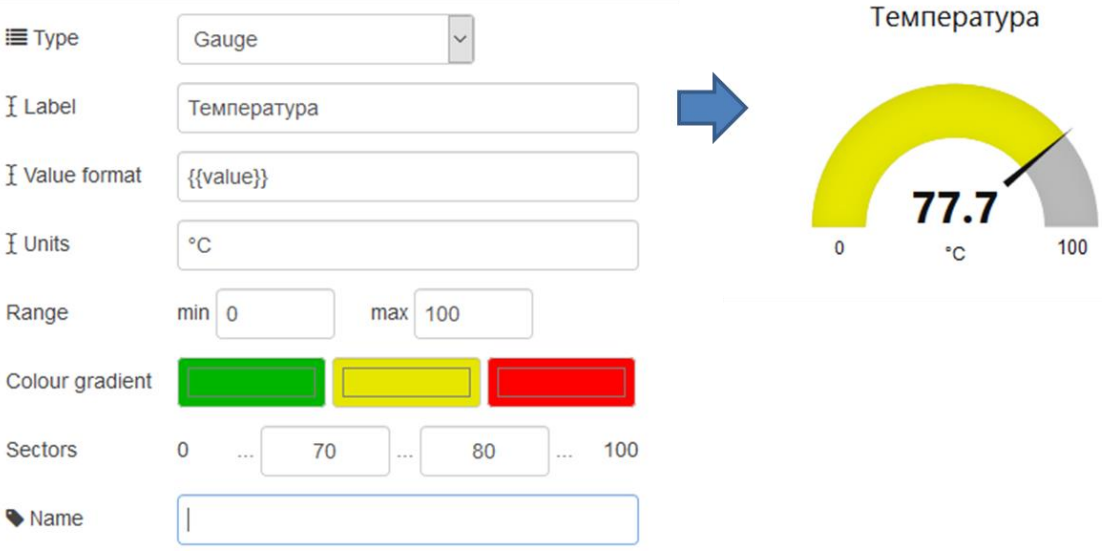
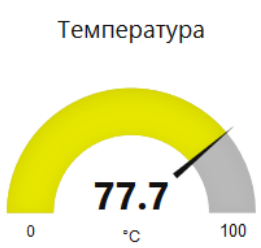
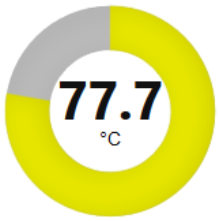
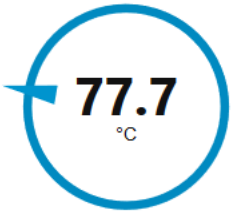
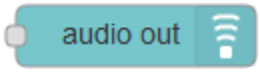


рис.23. Налаштування Gauge.

Віджет намагається перетворити `msg.payload` в числове значення і від форматувати його відповідно до означеного **Value Format**, використовуючи ангулярні фільтри. Можуть бути вказані кольори кожного з 3 секторів. Тип означує зовнішній вигляд індикатору. В таблиці вказаний приклад вигляду індикатору при тих же налаштуваннях, але різних типів.

Gauge	Donut	Compass	Level
			

Audio out (Вивід аудіо)



Відтворює аудіо або текст в мову (text to speech TTS).
Для роботи на веб-сторінці приладова панель має бути відкритою. Очікує, що `msg.payload` містить буфер файлу wav або mp3. Коли на вхід вузла поступає `msg.reset` зі значенням true, то відтворення поточного фрагмента звуку буде зупинено.

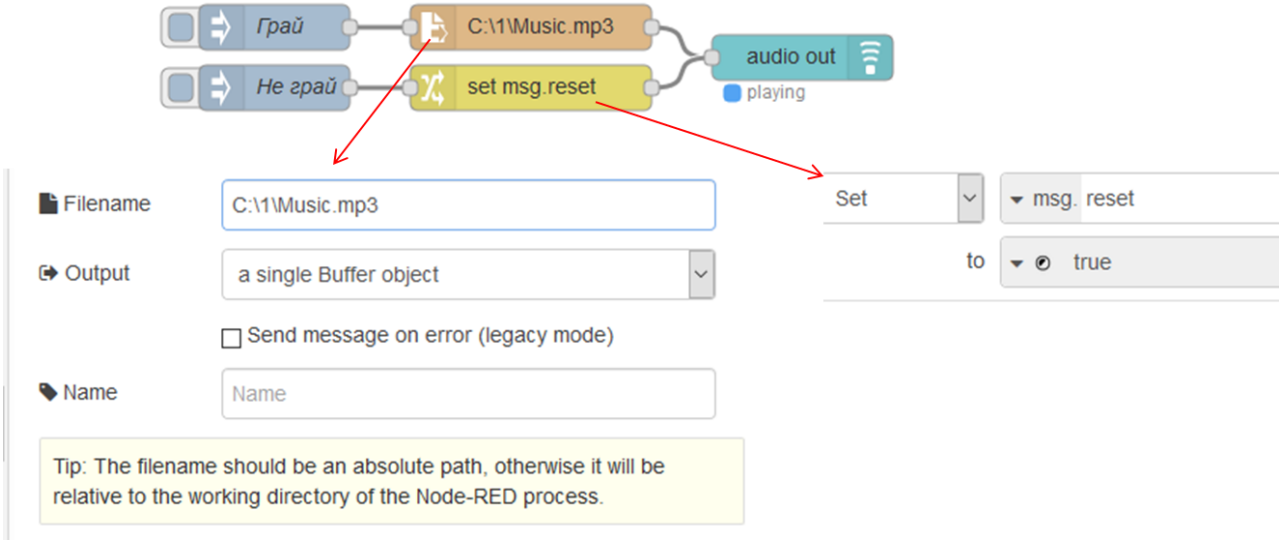


рис.24. Приклад фрагменту програми для керування відтворенням звукового файлу.

Якщо браузер має вбудовану підтримку тексту в мову, то msg.payload може також бути рядком, що читається вголос.

Статус вузла відображає поточний стан відтворення:
• started: почалося відтворення аудіо фрагмента.
• reset: відтворення аудіо фрагмента було скинуто (тобто зупинено до завершення).
Як тільки відтворення аудіо фрагменту буде завершено, стан вузла буде очищено автоматично.

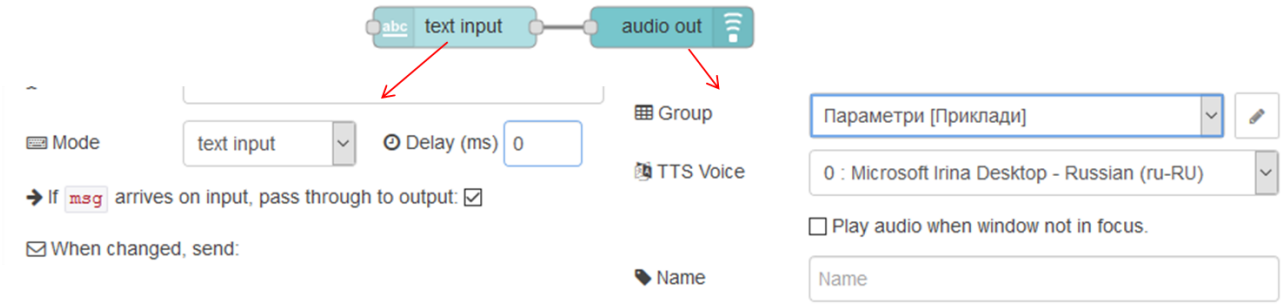
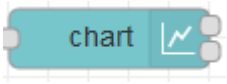


рис.25. Приклад фрагменту програми для проговорення тексту .

Chart (Діаграма)



Додає до інтерфейсу користувача діаграму з відображенням значень, що надходять на вхід. Це може бути лінійний графік від часу (time based line chart), стовпчикова діаграма (bar chart, вертикальна або горизонтальна) або кругова діаграма(pie chart).

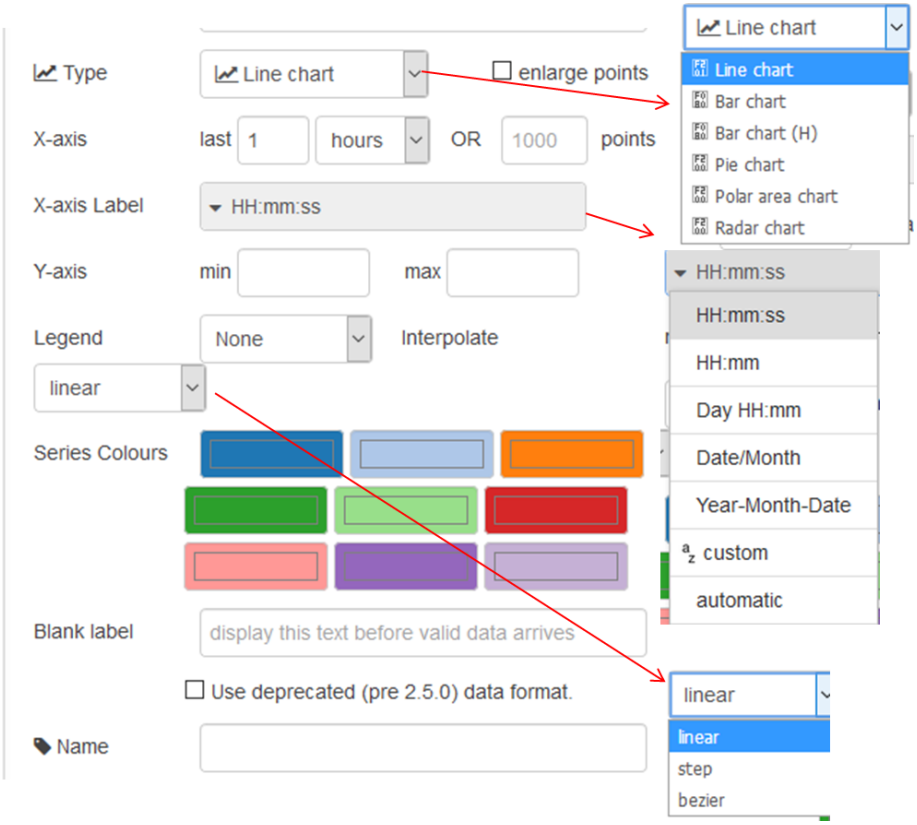


рис.26. Вікно налаштування вузла Chart.

Кожне значення `msg.payload` перетворюється в числове значення. Якщо перетворення не вдається, повідомлення ігнорується. Властивість максимального і мінімального значення осі Y (**Y** axis Min та Max) є опціональними, якщо вони не вказуються - застосовується авто-масштабування.

Вісь X для діаграм типу Line Chart означує часове вікно або максимальну кількість точок для відображення. Старі дані автоматично видаляються з графіка. Мітки осі можуть бути відформатовані за допомогою рядка Moment.js time formatted, відформатованого за часом.

На тій самій діаграмі можуть бути показані кілька послідовностей (series). Для цього використовуючи різне значення `msg.topic` для кожного вхідного повідомлення, колір кривих

означується в налаштуванні вузла і може бути зміненим. Для прикладу розглянемо фрагмент програми з використанням діаграм типу Line Chart, зображеної на рис.27.

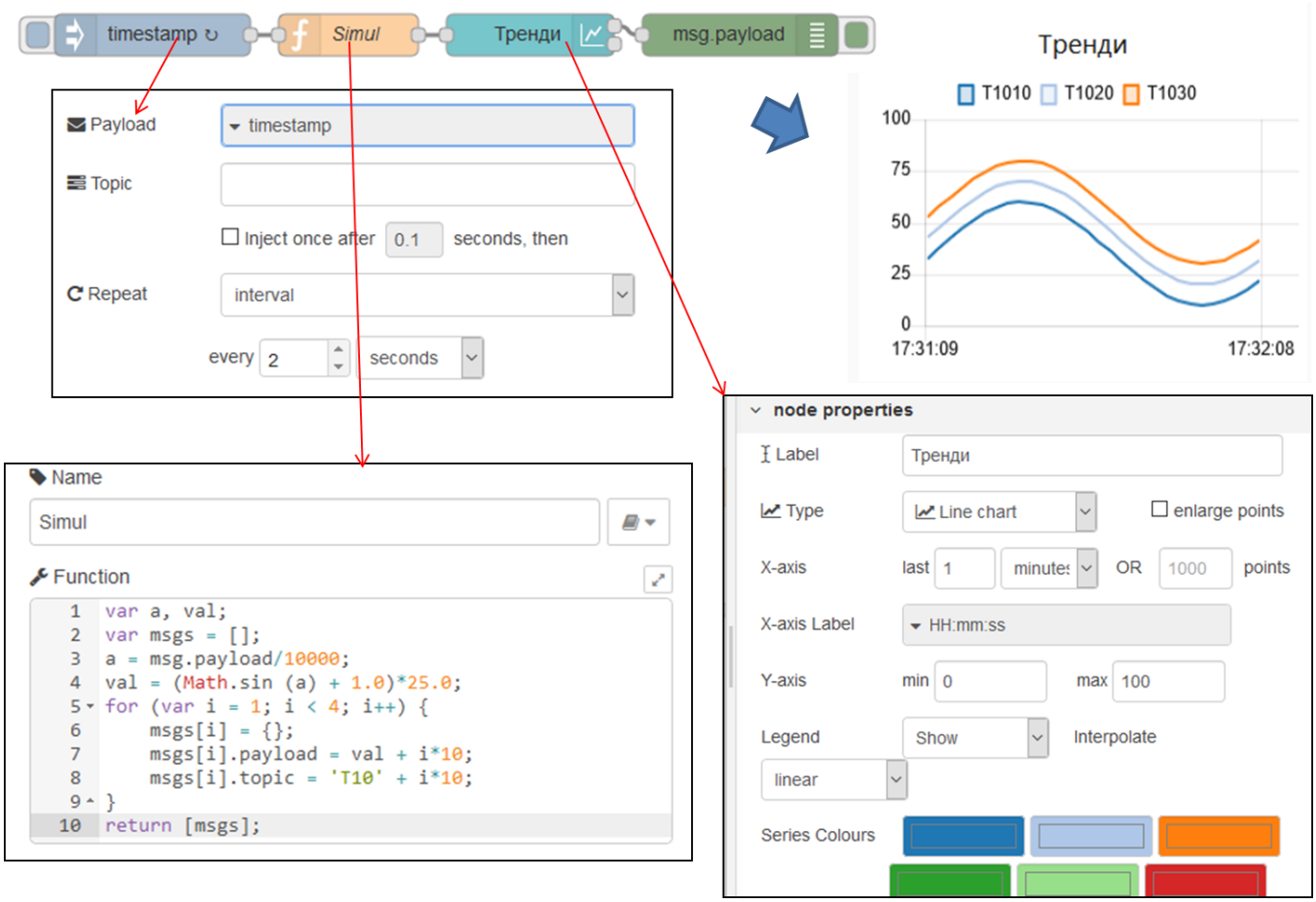


рис.27. Фрагмент програми та результат її роботи для використання Chart типу Line Chart.

Вузол Timestamp формує у вихідному повідомленні відмітку часу, яку вузол Simul (для імітації) обробляє як число для здобуття кута в радіанах для синусоїди. Отримане значення синусоїди (Math.sin) зміщується на +1.0 та множиться на масштабний коефіцієнт, щоб отримати значення від 0 до 50. Далі створюється масив з 3-х об'єктів msgs, для кожного з яких означується payload (кожне зміщене на 10 відносно val) та topic («T1010», «T1020», «T1030»). На вихід відправляється масив повідомлень, що формує їх послідовність. Таким чином вузол тренди отримує повідомлення з різними topic для яких формує графіки залежності від часу тривалістю 1 хвилину. На легенді видно назви цих графіків, тобто topic. Кольори вибираються з послідовності, яка означена в самому вузлі.

Вихід вузла містить масив стану діаграми, який, за необхідності, може бути збережений. Це може бути передано до вузла діаграми для повторного відображення збережених даних. Другий вихід вузла більше не повинен використовуватися і буде видалено в майбутньому випуску. Використовуйте вузол `control` для запуску відновлення, якщо це необхідно. Ось як виглядить вихідне повідомлення з попереднього прикладу (див.рис.28). При кожному оновленні вузла, на вихід в payload записується масив з одного елементу object, що містить три масиви об'єктів: «series», «data» та «labels». Для типу діаграми LineChart мають значення тільки «series» та «data». Масив «series» вказує на назви послідовностей (графіків), тобто на їх topic. Масив «data» - є масивом даних з трьох елементів для кожної послідовності (графіку), кожний з яких є масивом об'єктів для кожної точки на графіку : x – відмітка часу, y – значення.

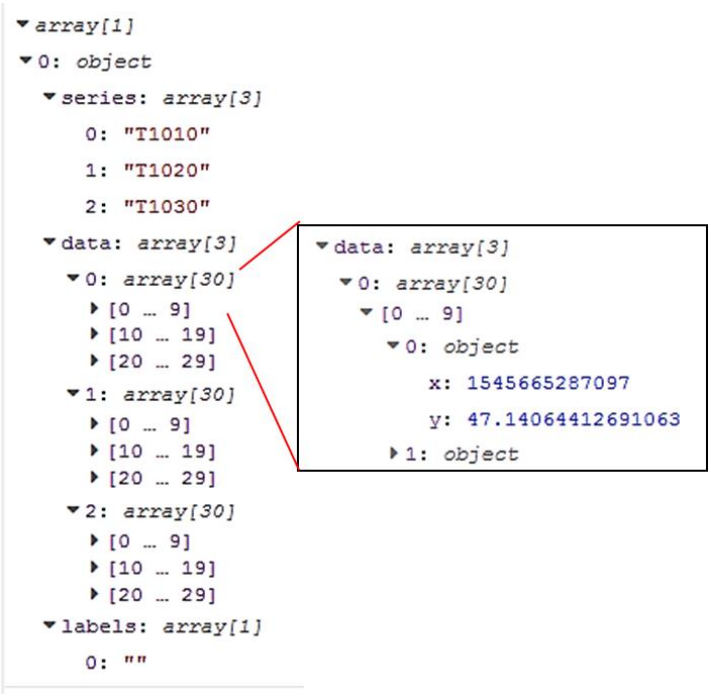


рис.28. Вигляд вихідного повідомлення з вузла Тренди з прикладу.

Згідно документації, за допомогою властивості `msg.label` для інших типів діаграм (не line chart) можна показати кілька стовпчиків (labels) тієї ж послідовності. Однак на практиці `msg.topic` використовується аналогічно як і `msg.label`. Якщо модифікувати наведений вище приклад, добавивши такі самі вузли chart але з іншими типами, то вийде наступний результат (див.рис. 29). Вихідне повідомлення з вузлів цього типу показані на рис.30. Як видно, замість масиву “series” на виході жаного вузла формується масив «labels».

Вхідне повідомлення `msg.payload` що містить пустий масив `[]` автоматично очищає діаграму. Дивіться [це посилання] для розуміння як наперед сформатовані дані можна вивести на готову діаграму. Інтерфейс показаний на рис.28.1

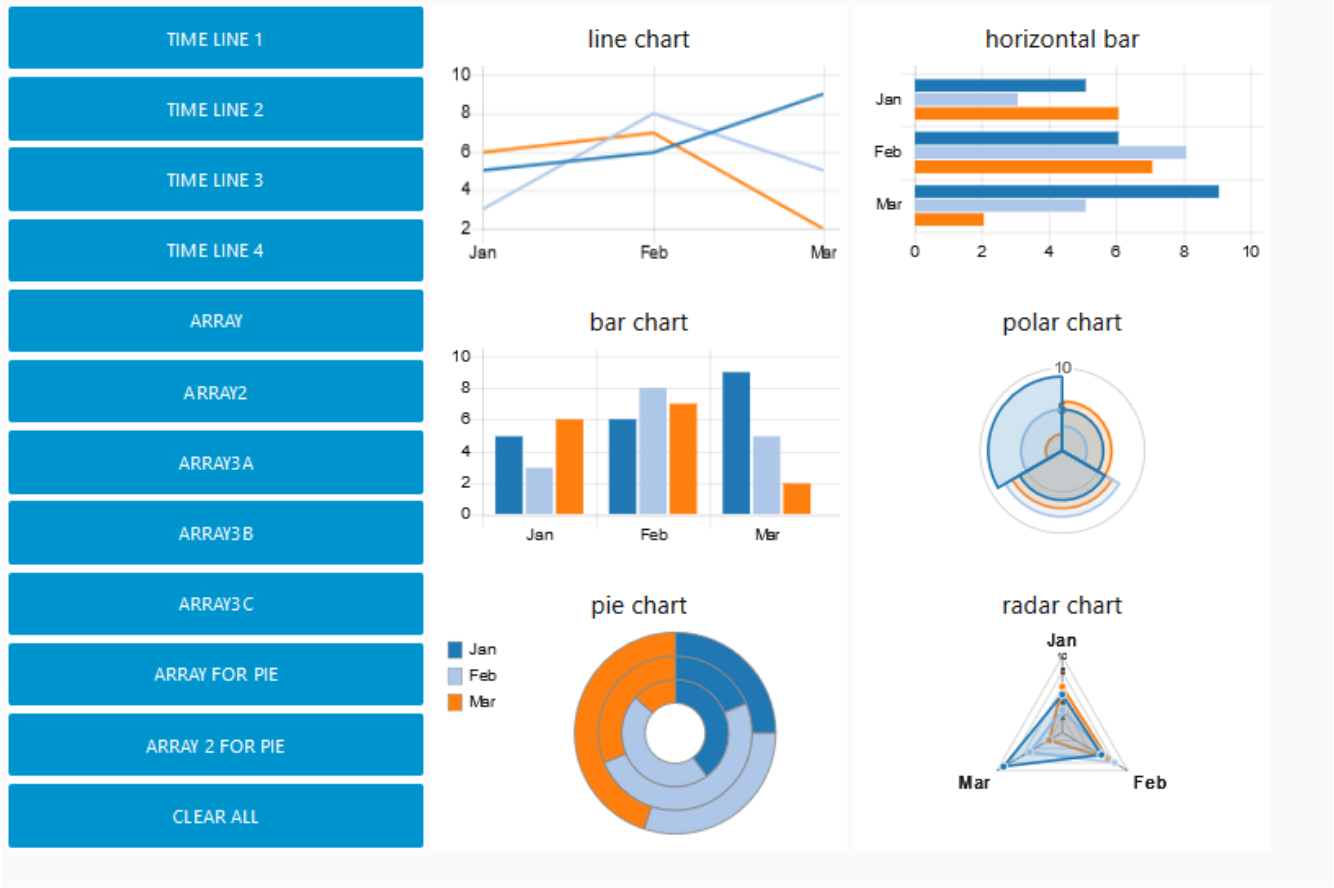


рис.28.1.

Поле **Blank label** використовується для відображення тексту на діаграмі, коли дані відсутні. Мітку над діаграмою можна змінювати властивістю повідомлення, виставивши її в поле label, наприклад `{{msg.topic}}`.

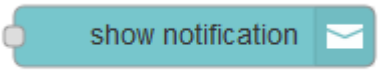
Див. також https://github.com/node-red/node-red-dashboard/blob/master/Charts.md

рис.29. Фрагмент програми та результат її роботи для використання Chart різних типів.

```
▼ array[1]
▼ 0: object
  ▼ series: array[1]
      0: ""
  ▼ data: array[1]
    ▼ 0: array[3]
        0: 40.78804397115536
        1: 50.78804397115536
        2: 60.78804397115536
  ▼ labels: array[3]
      0: "T1010"
      1: "T1020"
      2: "T1030"
```


рис.30. Вигляд вихідного повідомлення з вузла Pie Chart з прикладу.

Show notification (Повідомлення)



Показує `msg.payload` як спливаюче сповіщення або діалогове повідомлення з кнопками OK/Cancel.

Якщо не встановлюється додатковий колір підсвічування рамки в налаштуваннях, то він може бути встановлений динамічно за допомогою `msg.highlight`. Також можна налаштувати місце (верхній/нижній правий/лівий куток екрану) та тривалість повідомлення.

При вибору діалогового вікна (Ok/Cancel Dialog), вузол повертає рядок `msg.payload` в якому вказує натиснуто кнопку. Означення напису для другої кнопки (скасування) є необов'язковою, як і значення, що повертається `msg.topic`.

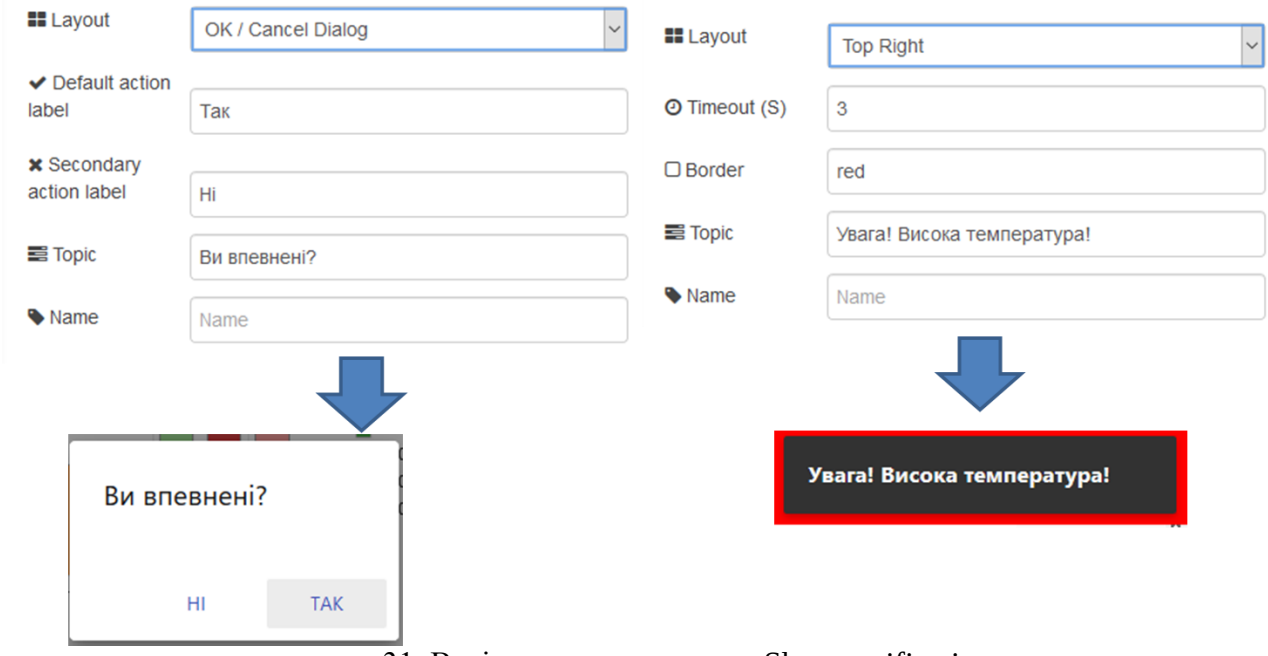
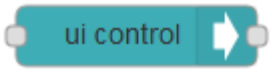


рис.31. Варіанти налаштування Show notification.

Ui control (Керування UI)



Дозволяє динамічне керування Dashboard.

За замовченням функція змінює активну вкладку. На вхід подається повідомлення де `msg.payload` має бути або об'єктом форми `{"tab":"my_tab_name"}`, або просто бути назвою вкладки або числовим індексом (з 0) вкладки або посилання, яка буде показане.

Якщо надіслати пусту назву вкладки "" оновиться поточна сторінка. Також можна надіслати "+1" для переключення на наступну вкладку та "-1" - на попередню.

Видимість окремих груп віджетів може керуватися payload, за подібною формою:
`{"group":{"hide":["tab_name_group_name_with_underscores"],"show":["reveal_another_group"], "focus":true}}`
, де `focus` є необов'язковим і призведе до прокручування екрана, щоб показати цю групу, якщо потрібно. Імена груп є ідентифікаторами груп і зазвичай формуються з назви закладок та назви групи, об'єднаної символом підкреслення, що замінює всі пробіли. Наприклад, на рис.32 показаний фрагмент програми, в якому використовується перемикач «Тренди» що перемикає видимість двох груп елементів «Група1» та «Група2», які знаходяться на вкладці «Приклади».

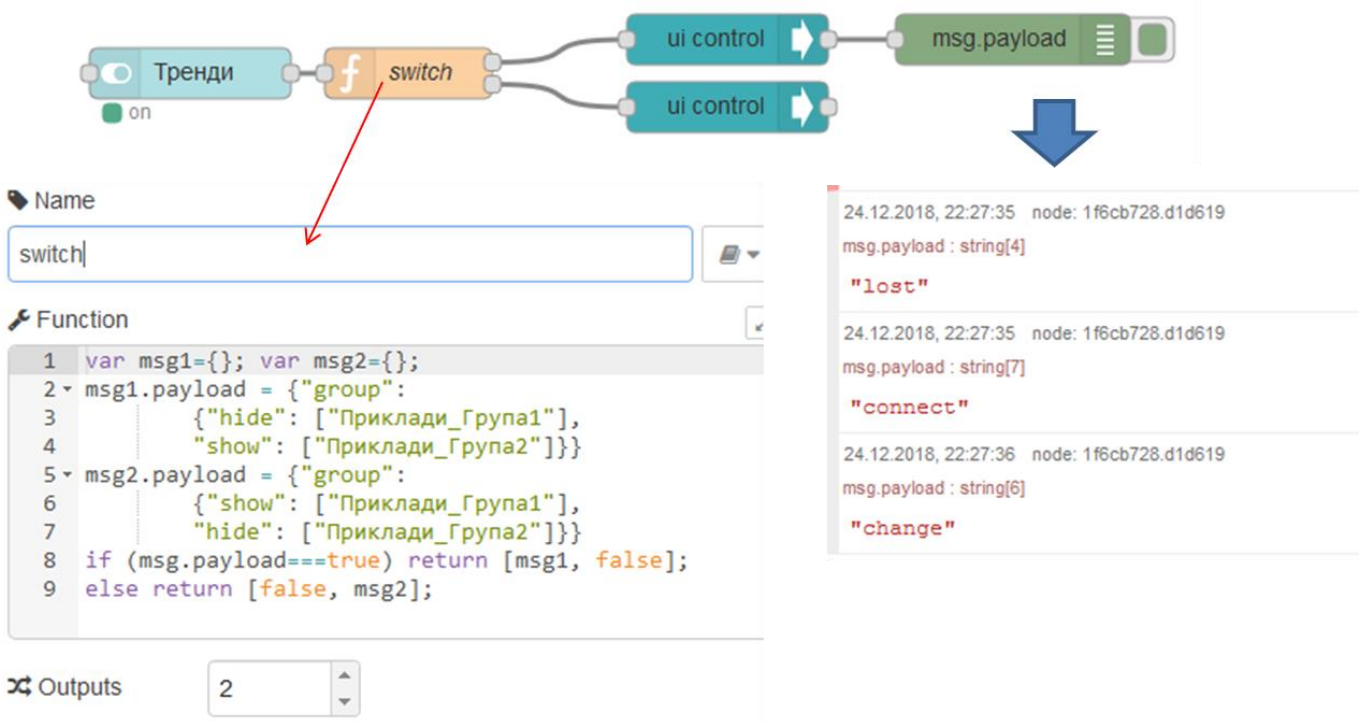


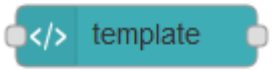
рис.32. Приклад використання Ui control для перемикання видимості груп.

Коли будь-який клієнт браузера підключається або втрачає зв'язок, цей вузол видає `msg`, що містить:

- `payload` - *connect* (підключення) або *lost* (втрата)
- `socketid` - ідентифікатор сокета (це буде змінюватися кожного разу, коли браузер перезавантажує сторінку).
- `socketip` - ip-адреса, з якої виникло з'єднання.

На рис.32 показаний приклад виведення повідомлень з вузла ui_control.

Template (Шаблон)



Цей підрозділ є «сирим» та неповним, тобто потребує обробки та доповнення. Зокрема потрібні для консультації спеціалісти по JavaScript та AngularJS.

Шаблонний віджет (template widget) може містити будь-які дійсні директиви html та Angular / Angular-Material.

Цей вузол може бути використаний для створення динамічного елементу інтерфейсу користувача, який змінює свій вигляд на основі вхідного повідомлення і може посилати повідомлення назад до Node-RED.

Наприклад:

```
<div layout="row" layout-align="space-between">
  <p>The number is</p>
  <p ng-style="{color: (msg.payload || 0) % 2 === 0 ? 'green' : 'red'}">
    {{(msg.payload || 0) % 2 === 0 ? 'even' : 'odd'}}
  </p>
</div>
```

Буде змінювати текст в залежності від парності числа, отриманого `msg.payload`. Він також змінить колір тексту на зелений, якщо число парне, або на червоний, якщо непарне.

У наступному прикладі показано, як встановити унікальний ідентифікатор для вашого шаблону, підібрати колір теми за замовчуванням і переглянути будь-яке вхідне повідомлення.

```
<div id="{{'my_'+$id}}" style="{{'color:'+theme.base_color}}">Some text</div>
<script>
(function(scope) {
  scope.$watch('msg', function(msg) {
    if (msg) {
      // Do something when msg arrives
      $("#my_"+scope.$id).html(msg.payload);
    }
  });
})(scope);
</script>
```

Шаблони, зроблені таким чином, можуть бути скопійовані і залишатися незалежними один від одного.

Відправка повідомлення:

```
<script>
var value = "hello world";
// or overwrite value in your callback function ...
this.scope.action = function() { return value; }
</script>
<md-button ng-click="send({payload:action()})">
    Click me to send a hello world
</md-button>
```

Will display a button that when clicked will send a message with the payload `'Hello world'`.

З'явиться кнопка, яка при натисканні надішле повідомлення з payload `'Hello world'`.

Використання `msg.template`:

Ви також можете визначити вміст шаблону за допомогою `msg.template`, наприклад, можна використовувати зовнішні файли.

Шаблон буде перезавантажено на вхід, якщо він був змінений.
Якщо в у полі Template `msg.template` присутній код, написаний, його буде проігноровано.

Наступні шрифтові значки також доступні: Material Design icons, Font Awesome icons, Weather icons.

Директиви AngularJS

Директиви AngularJS дозволяють розробнику модифікувати поведінку деяких елементів, чи описати власні елементи.

ng-app
: Оголошує елемент кореневим елементом застосунку, дозволяючи змінювати поведінку за допомогою спеціальних тегів.

ng-bind
: Змінює текст елемента на значення виразу. `<span ng-bind="name"></span>` відобразить значення змінної `name` всередині тегу `span`. Будь-які зміни змінної будуть миттєво відображені в DOM, де б змінна не використовувалась.

ng-init
: Ініціалізує/визначає дані/змінні вашого додатку.

ng-model
: Подібна до `ng-bind`, але дозволяє двостороннє зв'язування даних (зміни в DOM будуть змінювати змінну).

ng-class
: Дозволяє динамічно додавати та забирати класи елемента.: `<div class="{'active': activeDiv}"></div>`

ng-controller
: Вказує клас JavaScript контролера.

ng-repeat
: Створює кілька екземплярів елемента, для кожного об'єкта колекції.

ng-show & ng-hide
: Показують чи ховають елемент залежно від значення булевого виразу. Це досягається за допомогою задання в CSS атрибуту `display`.

ng-disabled

Встановлює атрибут `disabled` для елемента (кнопка, поле вводу, тощо), якщо вираз в середині ng-disabled вірний.

`<input ng-model='name' ng-disabled='name.length > 15'>` (встановить атрибут disabled для поля вводу при введенні рядка більше 15 символів)

ng-click/ngDblclick

Виконує описаний вираз при кліку/подвійному кліку на елемент. `<button ng-click='submitForm()'></button>` (Виконує функцію submitForm при кліку на кнопку)

ng-mousedown/ng-mouseup

Подібні до ng-click, спрацьовують при натисканні/відпусканні лівої кнопки миші на елементі.

ng-if

Видаляє чи створює елемент в DOM дереві. `<div ng-if='showBlock'>...</div>` Директива дуже подібна до директиви ng-show. Для того щоб запобігти втрат у швидкодії, рекомендується застосовувати директиву ng-show для приховування великої кількості елементів у зв'язку повільної роботи DOM дерева.

Про AngularJS
https://docs.angularjs.org/tutorial
http://javastudy.ru/angularjs/angularjs-hello-world-example/

http://javastudy.ru/angularjs/angularjs-filters/
https://metanit.com/web/angular/2.4.php
https://scotch.io/tutorials/all-about-the-built-in-angularjs-filters

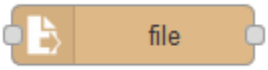
Створення своїх віджетів

https://github.com/node-red/node-red-dashboard/wiki/Creating-New-Dashboard-Widgets

Цей підрозділ є «сирим» та неповним, тобто потребує обробки та доповнення. Зокрема потрібні для консультації спеціалісти по JavaScript та AngularJS.

Вбудовані функції роботи з файлами

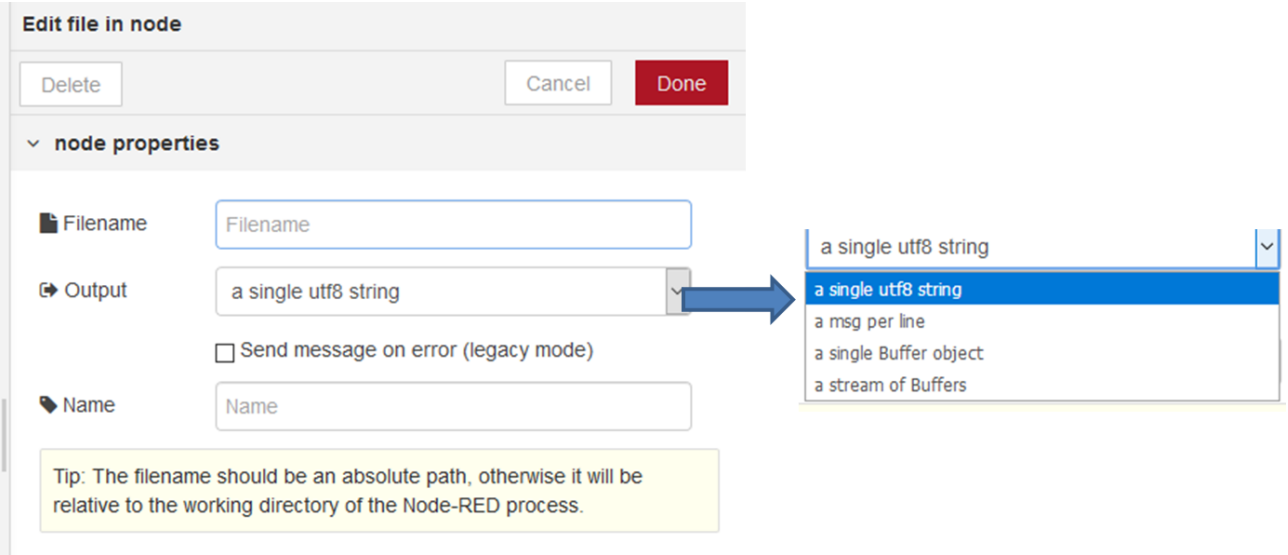
File in



Читає вміст файлу у вигляді рядку або бінарного буферу.
На вхід подається ім’я файлу (filename з типом string) для читання. Ім’я файлу також може бути вказано у конфігурації вузла
На вихід посилаються наступні властивості повідомлення:

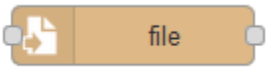
- payload (string | buffer) – вміст файлу як string або як binary buffer
- filename (string) – якщо не сконфігуровано у властивостях вузлу, ця властивість вказує на файл, що считувався
- error (object) – застаріле, якщо в налаштуваннях вузла активована опція «Send message on error (legacy mode)», коли вузол виявляє помилку при читанні файлу, то відправить повідомлення без `payload`, і в цій властивості `error` буде встановлені деталі помилки. Цей режим поведінки застарілий і не включений за умовчанням для нових екземплярів вузла. До версії Node-RED 0.17, якщо в цьому вузлі виникала помилка під час читання файлу, він відправляв повідомлення без `msg.payload` та формував `msg.error` для

деталізації помилки. Це застарілий режим поведінки для вузла, який в нових версіях не використовуються. Якщо потрібно, цей режим може бути явно включений в конфігурації вузла.
Наразі помилки слід сприймати та обробляти за допомогою вузла Catch.



Ім'я файлу має бути абсолютним шляхом, інакше він буде вважатися відносно робочого каталогу процесу Node-RED. У Windows може знадобитися вилучення сепараторів шляхів, наприклад: `\\Users\\myUser`.
За бажанням, вихід з вузла для текстового файлу можна розділити на рядки (a msg per line), виводячи одне повідомлення на рядок. Зчитування з бінарного файлу можна розділений на менші буферні фрагменти (stream of Buffers), розмір фрагменту залежить від операційної системи, але зазвичай 64k (Linux / Mac) або 41k (Windows).
Якщо зчитаний файл розбивається на кілька частин-повідомлень, кожне з них матиме властивість `parts`, утворюючи цілісну послідовність повідомлень.

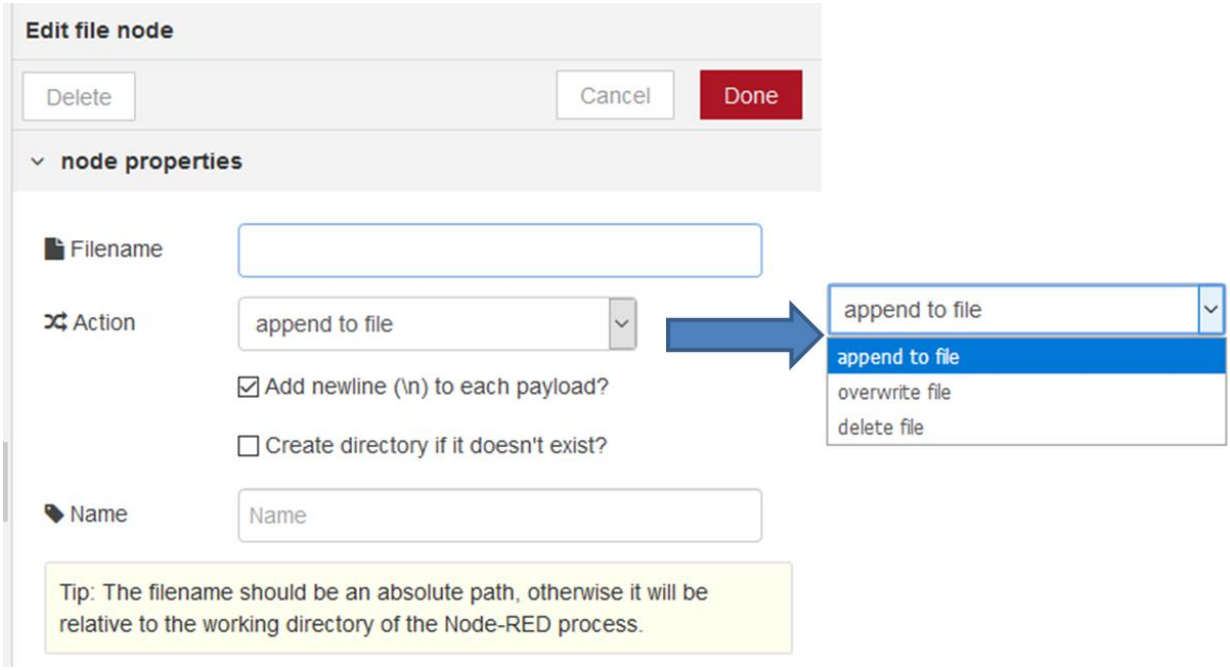
File



Записує `msg.payload` у файл, додавши його до кінця або замінюючи існуючий вміст. Крім того, вузол може видалити файл.

На вхід можна подати значення мені файлу (filename з типом string) який необхідно змінити. Після завершення запису, вхідне повідомлення надсилається на вихідний порт.

Кожне payload з повідомлення буде додано до кінця файлу. За необхідності можна додавати в кінець кожного повідомленнями символ нового рядку (\ n). Якщо використовується `msg.filename`, файл буде закрито після кожного запису. Для кращої продуктивності використовуйте фіксоване ім'я файлу. Вузол може бути налаштований на перезапис всього файлу, а не на додавання до нього. Наприклад, при складанні бінарних даних у файл, таких як зображення, слід використовувати параметр “overwrite file. Крім того, цей вузол може бути налаштований на видалення файлу (параметр “delete file”).



fs-ops базові операції з файлами

Опис бібліотеки

У бібліотеці представлені вузли Node Red для виконання операцій з файловою системою.

Ці вузли є обгорткою навколо багатьох функцій у бібліотеці файлових систем Node.js, яка, в свою чергу, є обгорткою навколо стандартних функцій файлів POSIX.

Ці вузли та функції забезпечують функції:

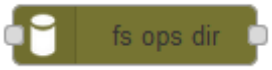
fs-ops-move - переміщення або перейменування файлів і каталогів
fs-ops-copy - копіювання або посилання на файли
fs-ops-delete - видалення файлів або каталогів
fs-ops-access - перевірка наявності та доступності файлу або каталогу
fs-ops-size - отримання розміру файлу або каталогу в байтах
fs-ops-link - визначає, чи є файл посиланням і повертає файл, на який він посилається.
fs-ops-type - визначає тип файлу - звичайний, каталог, символ або спеціальний
fs-ops-dir - отримує масив імен файлів і каталогів у каталозі
fs-ops-mkdir - створює новий каталог
fs-ops-mktmpdir - створює новий каталог із випадковим унікальним ім'ям

Такі параметри, як шлях і ім'я файлу, можуть бути отримані з рядків, повідомлень, потоків або глобальних властивостей. Аналогічно, результати можуть зберігатися в повідомленні, потоці або глобальній властивості.

fs-ops-dir можна використовувати для вилучення списку файлів за допомогою фільтра, а потім передавати цей список до інших вузлів fs-ops для виконання об'ємних операцій.

Загальний принципи розробки полягає в тому, що кожен вузол передаватиме повідомлення, якщо дія буде успішною, інакше вона викине виняток і скине повідомлення.

dir (перегляд директорії)



Дає список файлів в каталозі файлової системи хоста. В якості конфігураційних параметрів вказується:

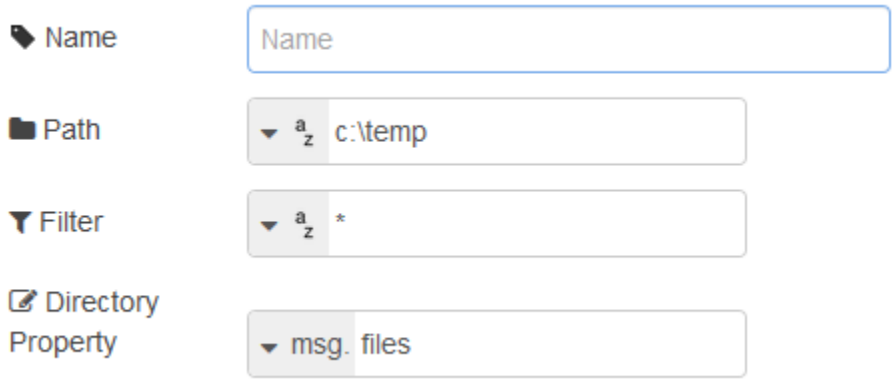
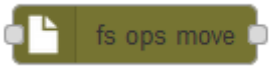


рис.1.

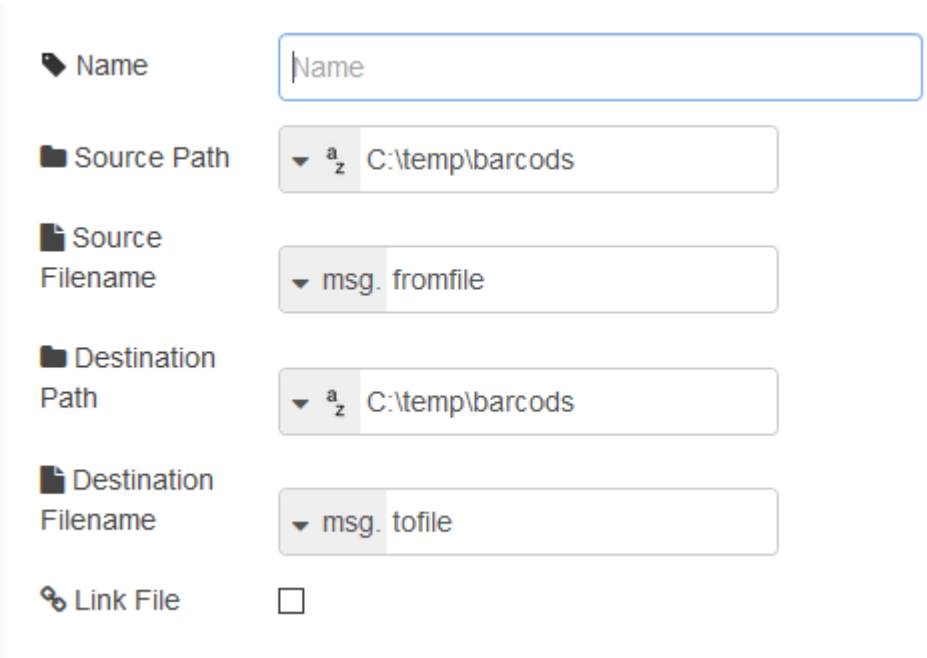
source path (string) – шлях директорії
filter (string) – фільтр імені файлу
directory (property) - властивість, в якій будуть записані імена файлів

На вихід подаються імена файлів, що зберігаються в масиві в заданій властивості.

move (переміщення/перейменування файлів)



Переміщує або перейменовує файли в локальній файловій системі. В якості конфігураційних параметрів вказується:
name (string) – ім’я вузла що відображається в редакторі
source path (string) – шлях директорії, що вміщує файл для переміщення/перейменування
source filename (string) – ім’я файлу що необхідно перемістити/перейменувати
destination path (string) – шлях директорії куди буде переміщено/перейменовано файл
destination filename (string) – нове ім’я файлу
link file (Boolean) – посилання до файлу призначення



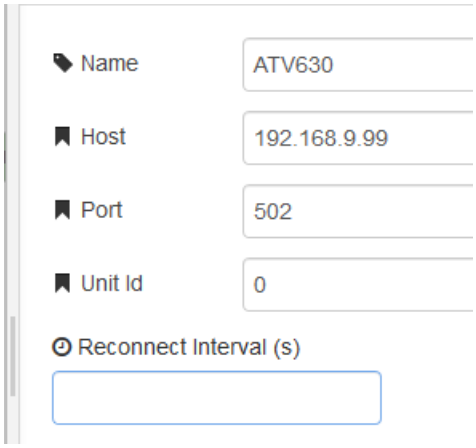
Увага: цей вузол буде завжди переписувати файл призначення.

Якщо встановлено опція *Link Files*, файл призначення буде символічним посиланням на вихідний файл. - Застаріле: Замість цього використовуйте вузол File Copy.
Повідомлення виконується і передається без змін, якщо переміщення/перейменування є успішним.

MODBUS

Про функціонування протоколу Modbus та Modbus TCP/IP ви можете дізнатися з відкритого дистанційного курсу «Промислові мережі».

Modbus-tcp Server (Config)

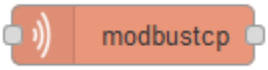


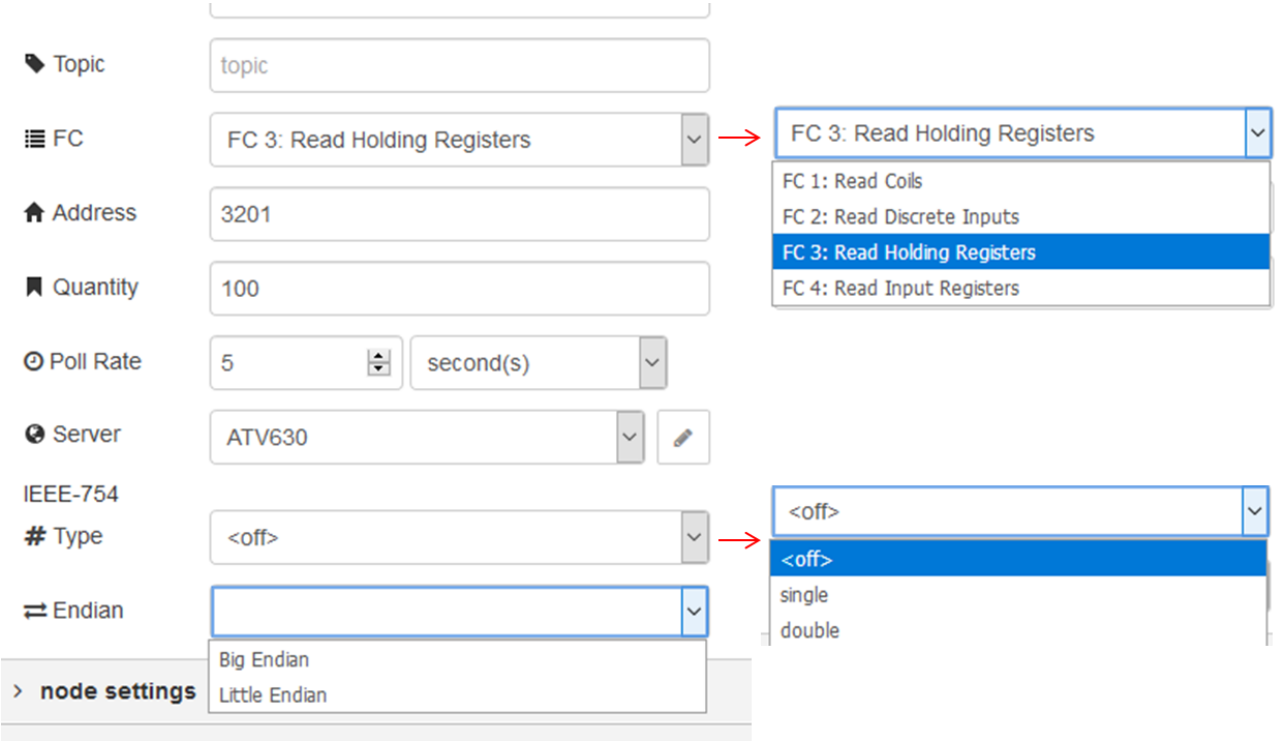
node-red-contrib-modbustcp

Конфігураційний вузол MODBUS-TCP Server використовується для вказівки серверу, з якого буде відбуватися читання/запис. Усі поля відповідають згідно призначенню протоколу.

Reconnect Interval дозволяє MODBUSTCP підключенням автоматично пере підключатися по TCP після вказаного інтервалу. Якщо автоматичне підключення не вимагається – залишити порожнім.

Modbus-tcp Read





Підключається до Modbus TCP server для зчитування змінних з вказаною періодичністю.

Підтримуються наступні функції:

- FC 1: Read Coils
- FC 2: Read Discrete Inputs
- FC 3: Read Holding Registers
- FC 4: Read Input Registers

Вибирається необхідна функція (FC), стартова адреса зміщення coil/input/register (0:65535) кількість, вказується періодичність зчитування (необхідно ставити більше нуля!).

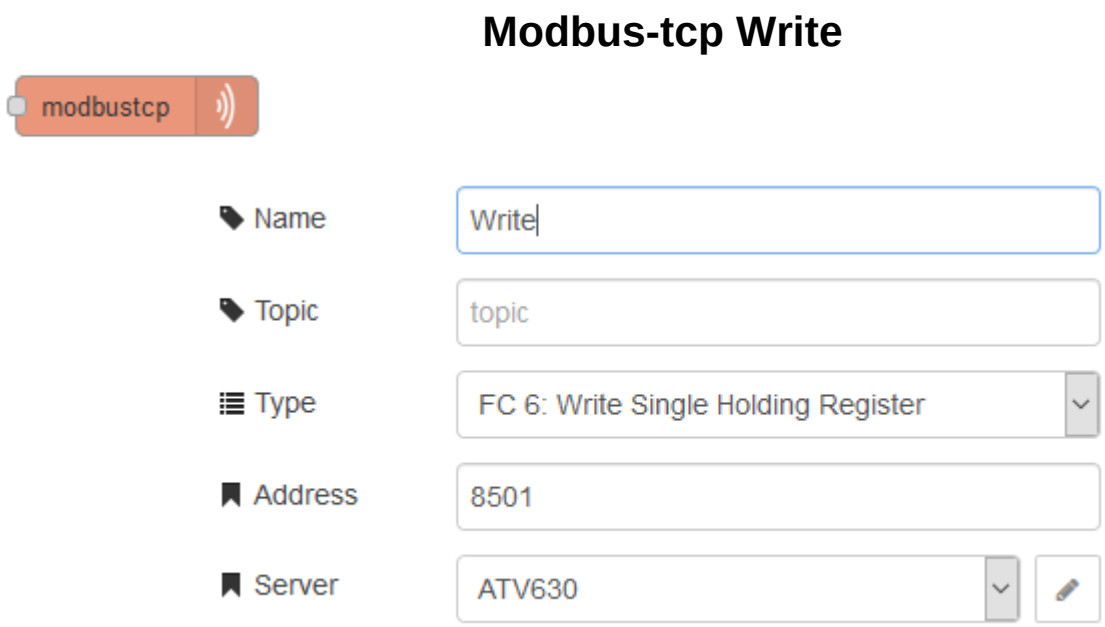
В полі IEEE-754 можна вибрати формат IEEE 754 Single або Double. Також вибирається порядок регістрів в них - Big або Little endian.

Входи:
Можна використовувати на вході один або більше значень payload для ініціювання зчитування окрім заданого періодичністю. Наприклад, можна використати вузол inject для вприскування JSON payload подібного до наступного:

```
{
    "name": "HoldingsDblBig",
    "topic": "Topic1",
    "address": 0,
    "quantity": 4,
    "dataType": "FC3",
    "interval": 3000,  // in ms
    "ieeeBE": true,
    "ieeeType": "double"
}
```

Виходи:
Вихід **msg** вміщує **msg.payload** з масивом прочитаних coils/inputs/registers.

msg.settings також вміщує додаткову інформацію про запит, що був використаний для генерування зчитування, включаючи **msg.settings.name**.



Підключається до TCP server для запису **msg.payload** в coil або register.

Підтримуються функції:

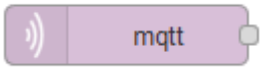
- FC 5: Write Single Coil
- FC 6: Write Single Holding Register
- FC 15: Write Multiple Coils
- FC 16: Write Multiple Holding Registers

У налаштуванні вибирається функція (FC), стартова адреса coil/register (0:65535).

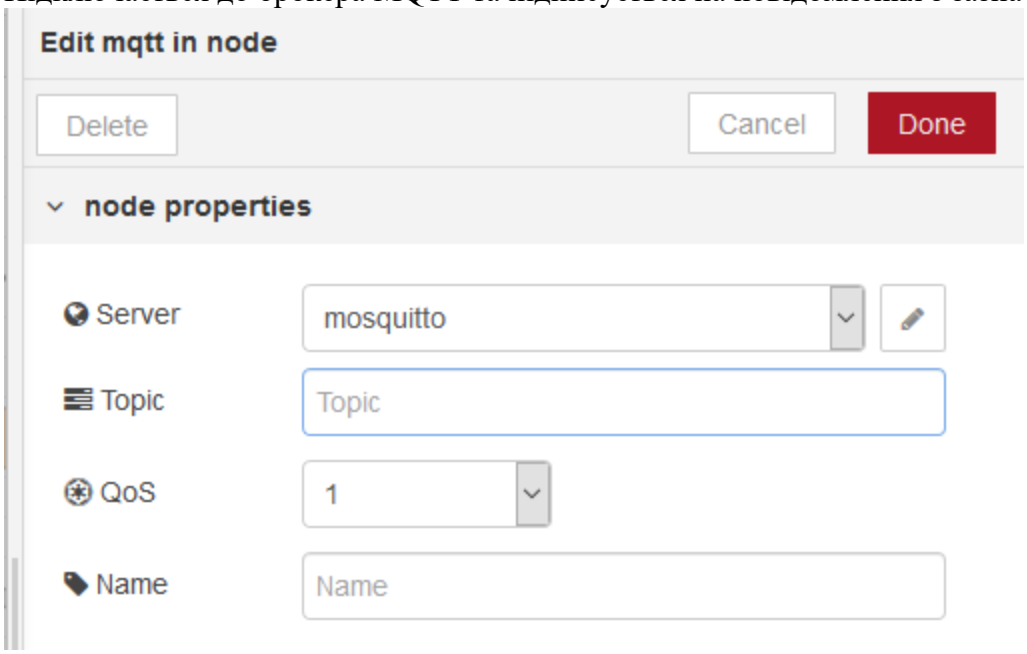
Для FC 5 у **msg.payload** вказується число або строкове значення 0 або 1. Для FC 6 у **msg.payload** повинна бути число або рядок між 0:65535. Для FC15, **msg.payload** повинен бути масивом чисел або рядків 0 або 1. Для FC 16, **msg.payload** повинен бути масивом чисел або рядків зі значеннями між 0:65535.

MQTT

Mqtt in



Підключається до брокера MQTT та підписується на повідомлення з зазначеної теми.



На виході вузла повідомлення містить наступні властивості:

- payload (string | buffer) – корисне навантаження у вигляді рядку або бінарного буферу
- topic (string) - MQTT topic використовується «/» як ієрархічний розділювач
- qos (number) – 0..2 (0, fire and forget - 1, at least once - 2, once and once only)
- retain (Boolean) – показує, що повідомлення було збережене і може бути старим

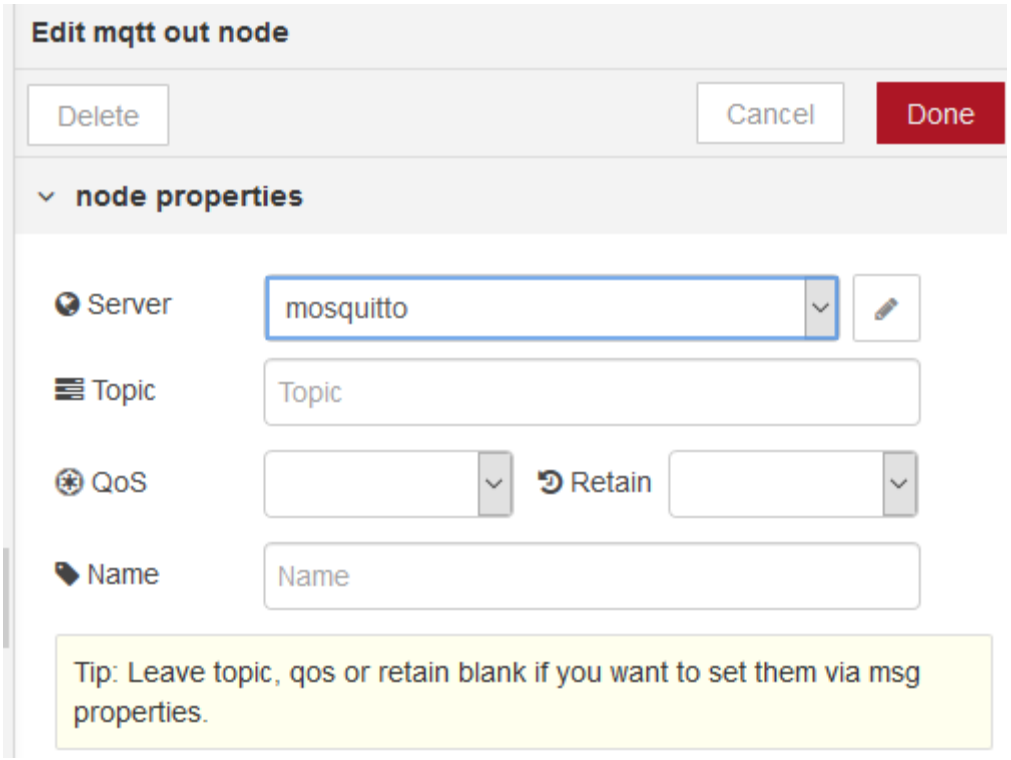
Тема підписки може включати підстановки MQTT, «+» для одного рівня, «#» для декількох рівнів. Цей вузол вимагає встановлення зв'язку з брокером MQTT, який налаштовується, натиснувши піктограму олівця.
За необхідності кілька вузлів MQTT (у вхідному або вихідному режимі) можуть мати доступ до одного і того ж брокеру.

Mqtt out

Підключається до брокера MQTT та публікує повідомлення.
В якості вхідного значення приймає наступні властивості повідомлень:

- payload (string | buffer) - більшість користувачів віддають перевагу простим текстовим корисним навантаженням, але також можуть бути опубліковані двійкові буфери.
- topic (string) - MQTT тема (topic) для публікації

- qos (number) – 0..2 (0, fire and forget - 1, at least once - 2, once and once only)
- retain (boolean) – виставити в true для збереження повідомлення на брокері



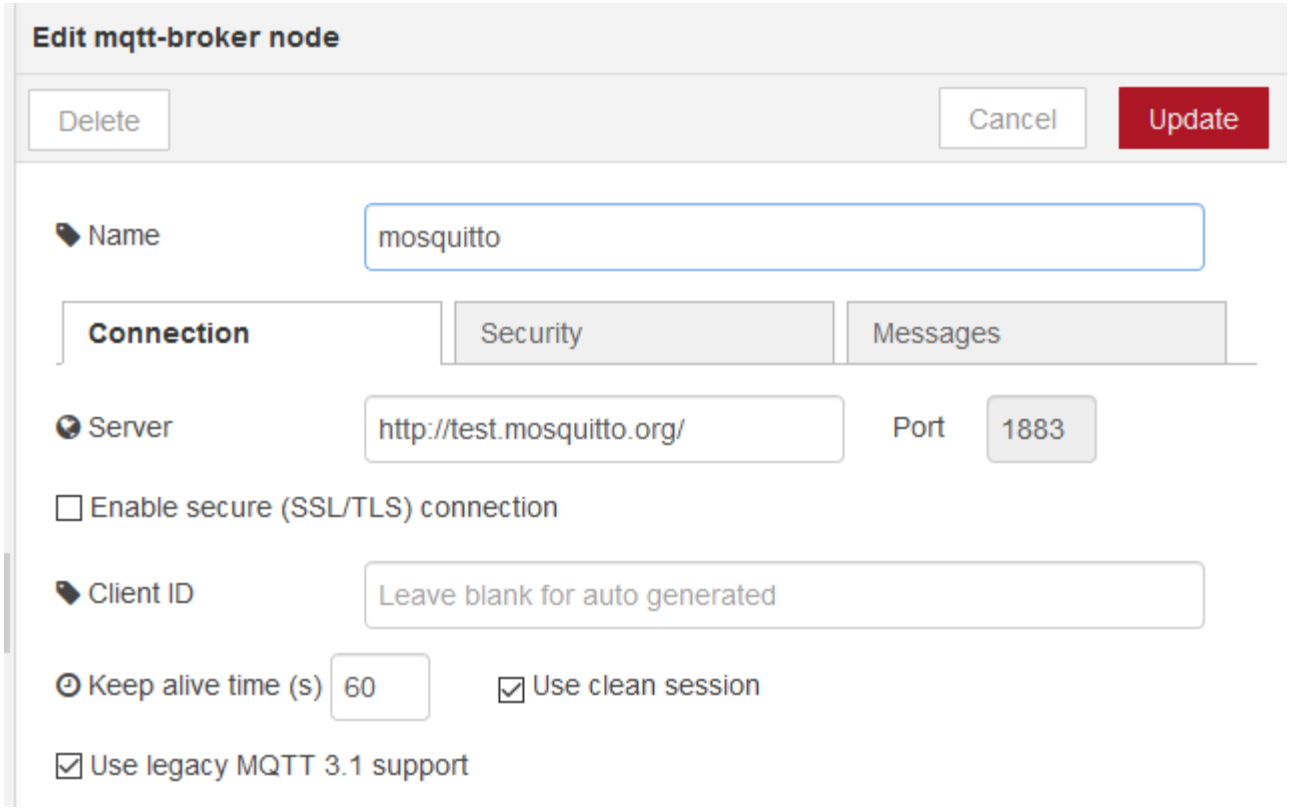
`msg.payload` використовується як корисне навантаження опублікованого повідомлення. Якщо він містить об'єкт, його буде перетворено в рядок JSON, перш ніж буде надіслано. Якщо він містить бінарний буфер, повідомлення буде опубліковано як-є.

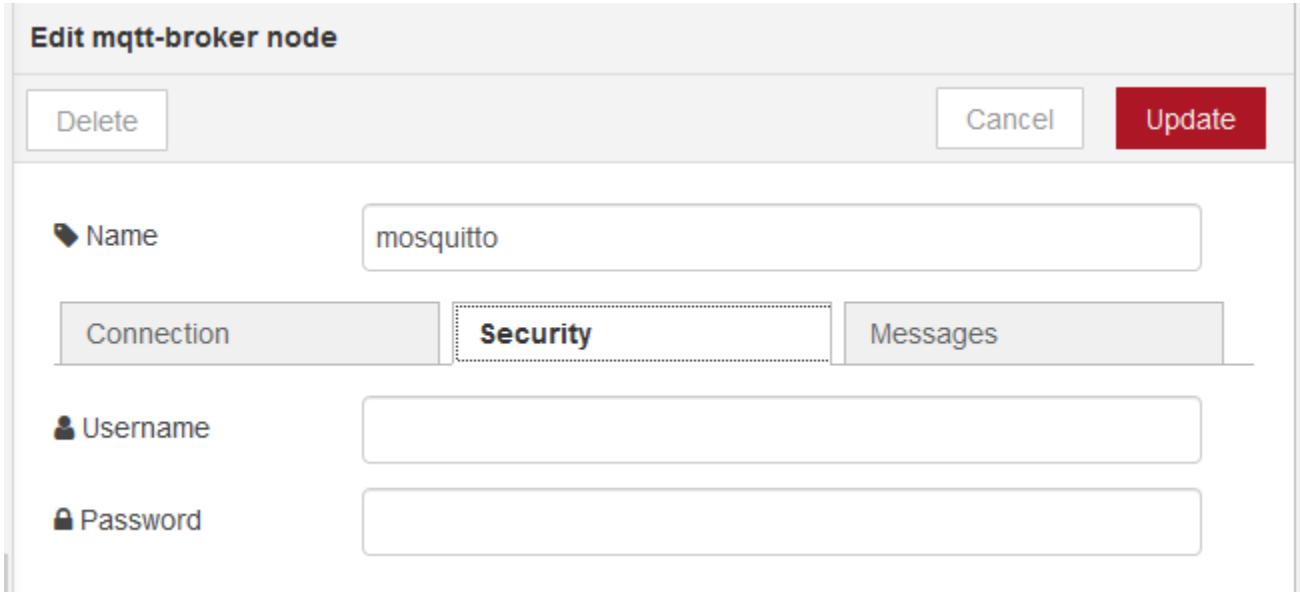
Використовувана тема може бути налаштована в вузлі або, якщо залишена порожньою, може бути встановлена за допомогою `msg.topic`.
Аналогічно, QoS та значення опції `retain` можуть бути налаштовані в конфігурації вузлу або, якщо вони залишаються порожніми, встановлюються відповідно `msg.qos` та `msg.retain` відповідно. Щоб очистити раніше збережену тему від брокера, надішліть порожнє повідомлення на цю тему за допомогою встановленого прапора.

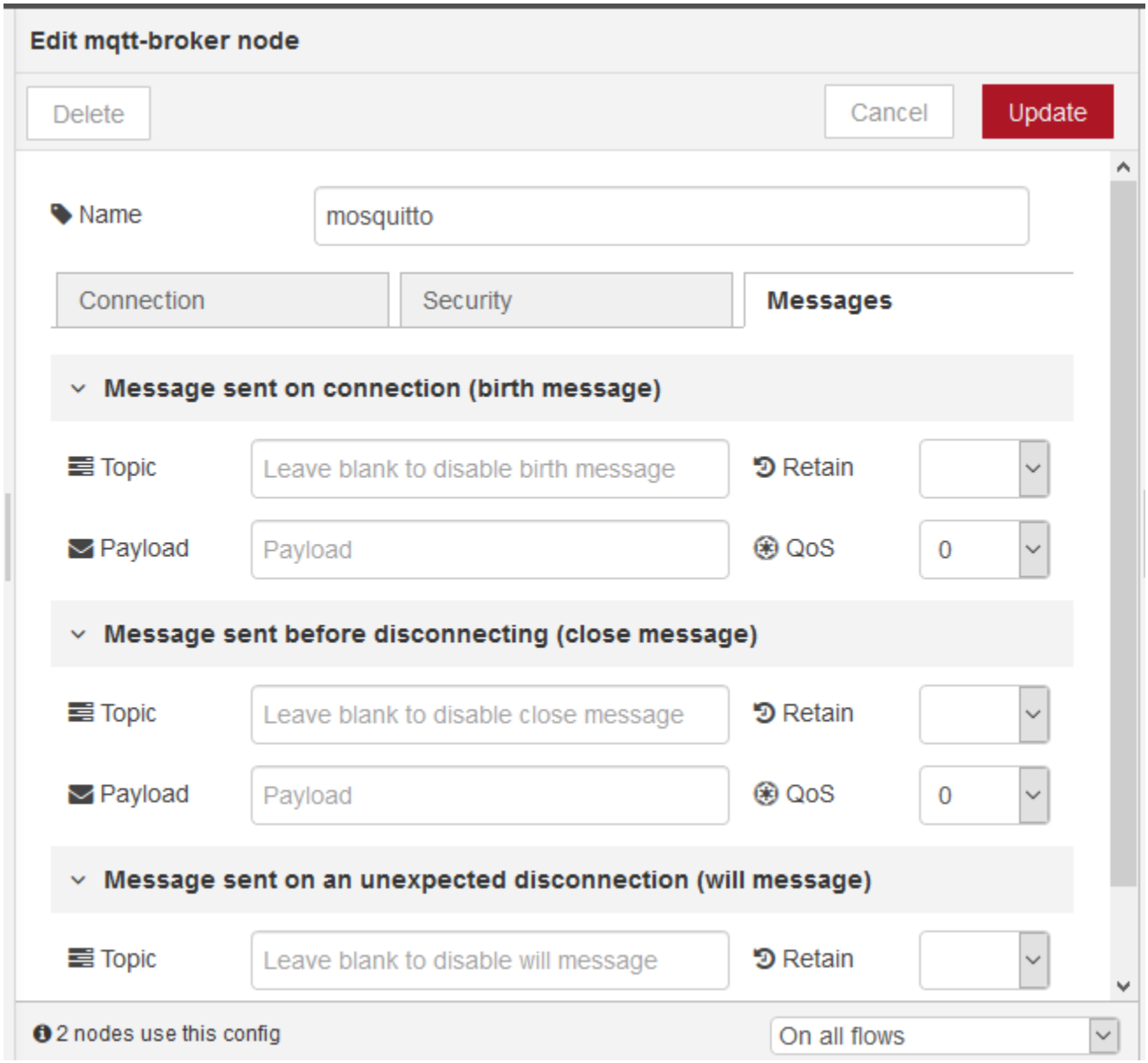
Цей вузол вимагає встановлення зв'язку з брокером MQTT. Це налаштовується, натисканням піктограми олівця.
За необхідності кілька вузлів MQTT (у вхідному або вихідному режимі) можуть мати доступ до одного і того ж брокеру.

Mqtt-broker (config)

Для налаштування брокерів MQTT є конфігураційні вузли Mqtt-broker. В налаштуваннях вузлів вказуються типові налаштування брокеру.


Edit mqtt-broker node
Delete
Cancel
Update
Name
mosquitto
Connection
Security
Messages
Server
http://test.mosquitto.org/
Port
1883
Enable secure (SSL/TLS) connection
Client ID
Leave blank for auto generated
Keep alive time (s)
60
Use clean session
Use legacy MQTT 3.1 support


Edit mqtt-broker node
Delete
Cancel
Update
Name
mosquitto
Connection
Security
Messages
Username
Password

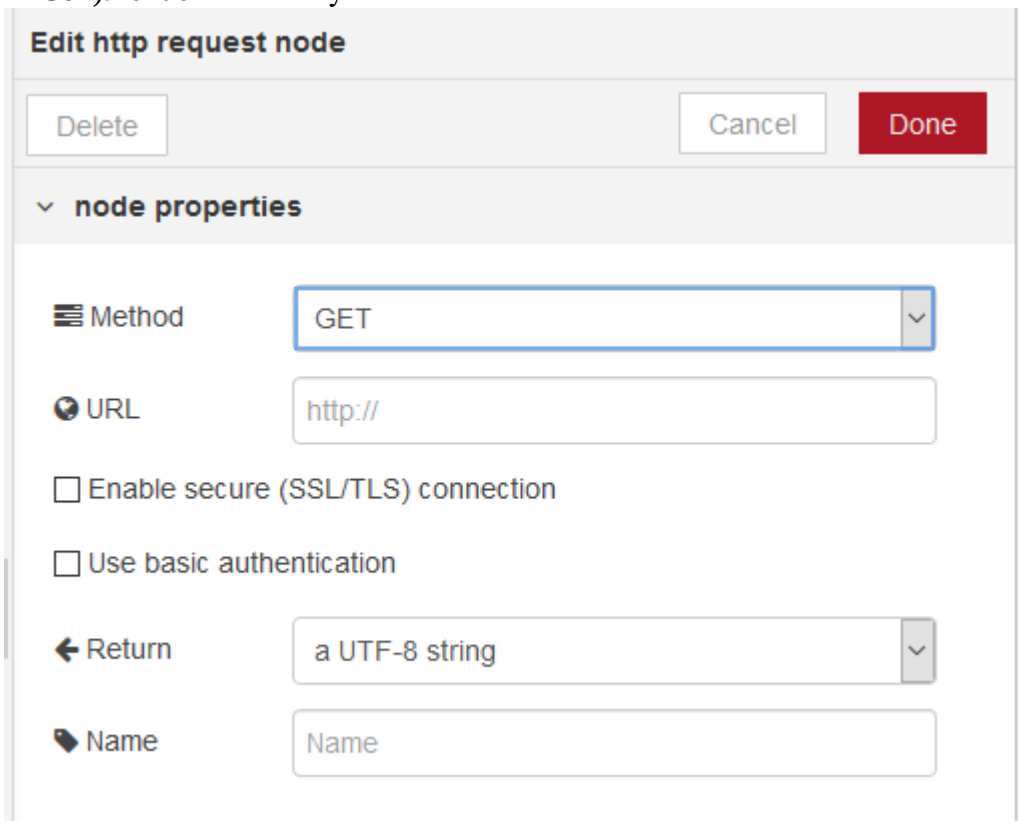

Edit mqtt-broker node
Delete
Cancel
Update
Name
mosquitto
Connection
Security
Messages
Message sent on connection (birth message)
Topic
Leave blank to disable birth message
Retain
Payload
Payload
QoS
0
Message sent before disconnecting (close message)
Topic
Leave blank to disable close message
Retain
Payload
Payload
QoS
0
Message sent on an unexpected disconnection (will message)
Topic
Leave blank to disable will message
Retain
2 nodes use this config
On all flows

HTTP

HTTP requests

Відправляє запити HTTP і повертає відповідь на нього. В якості вхідного значення приймає наступні властивості повідомлень:

- url (string) – якщо не сконфігуроване в вузлі, ця опціональна властивість висталвяє url для запиту.
- method (string) - якщо не сконфігуроване в вузлі, ця опціональна властивість виставляє метод HTTP для запиту. Повинно бути `GET`, `PUT`, `POST`, `PATCH` бо `DELETE`.
- headers (object) – виставляє HTTP заголовки в запиті
- cookies (object) – якщо вказані, можуть бути використані для відправки куків з запитом
- payload – виставляє тіло для запиту
- rejectUnauthorized – якщо виставлено в `false` дозволяє робити запити на сайти https, які використовують сертифікати, які підписуються самостійно
- followRedirects – якщо виставлено в `false` запобігає наступним перенаправленням (HTTP 301). `true` за замовчуванням



На виході формує:

- payload (string | object | buffer) – тіло відповіді. Вузол може бути налаштований так, щоб повернути тіло у вигляді string, спробувати розпарсити його як рядок JSON або залишити його у вигляді двійкового буфера.
- statusCode (number) - код стану відповіді або код помилки, якщо запит не може бути завершений.
- headers (object) – об’єкт, що містить заголовки відповідей

- responseUrl (string) - у випадку, якщо під час обробки запиту відбулися будь-які перенаправлення, це властивість є останньою адресою, що переадресовується. В іншому випадку це URL оригінального запиту.
- responseCookies (object) - якщо відповідь включає файли cookie, ця властивість є об'єктом пар імені/значення для кожного cookie.

Якщо сконфігуровано у вузлі, властивість URL може містити теги mustache-style. Вони дозволяють створювати URL, використовуючи значення вхідного повідомлення. Наприклад, якщо URL-адресу встановлено `example.com/{{{topic}}}` в це місце буде автоматично додано `msg.topic`. Використання {{{...}}} запобігає вилученню mustache із символів на зразок /& і т.д.

Примітка: Якщо запускається за проксі-сервері, необхідно встановити стандартну змінну середовища `http_proxy=...` і перезапустити Node-RED.

Для того щоб використовувати більше одного з таких вузлів в тому самому потоці, необхідно дотримуватися властивості `msg.headers`. Перший вузол встановить цю властивість з заголовками відповіді. Тоді наступний вузол буде використовувати ці заголовки для свого запиту - це звичайно не правильно. Якщо властивість `msg.headers` залишається незмінною між вузлами, вона буде ігноруватися другим вузлом. Щоб встановити користувальницькі заголовки, `msg.headers` слід спочатку видалити або скинути порожнім об'єктом: `{}`.

Властивість `cookies`, передана вузлу, повинна бути об'єктом з парої імя/значення. Значення для встановлення значення cookie може бути string, або об'єктом з єдиною властивістю `value`. Будь-які файли cookie, повернені запитом, передаються назад у властивості `responseCookies`.

Якщо `msg.payload` є Object, вузол буде автоматично встановлювати тип контенту запиту в `application/json` і кодувати тіло відповідним чином.

Для кодування запиту як форму даних `msg.headers["content-type"]` буде встановлюватися як `application/x-www-form-urlencoded`.

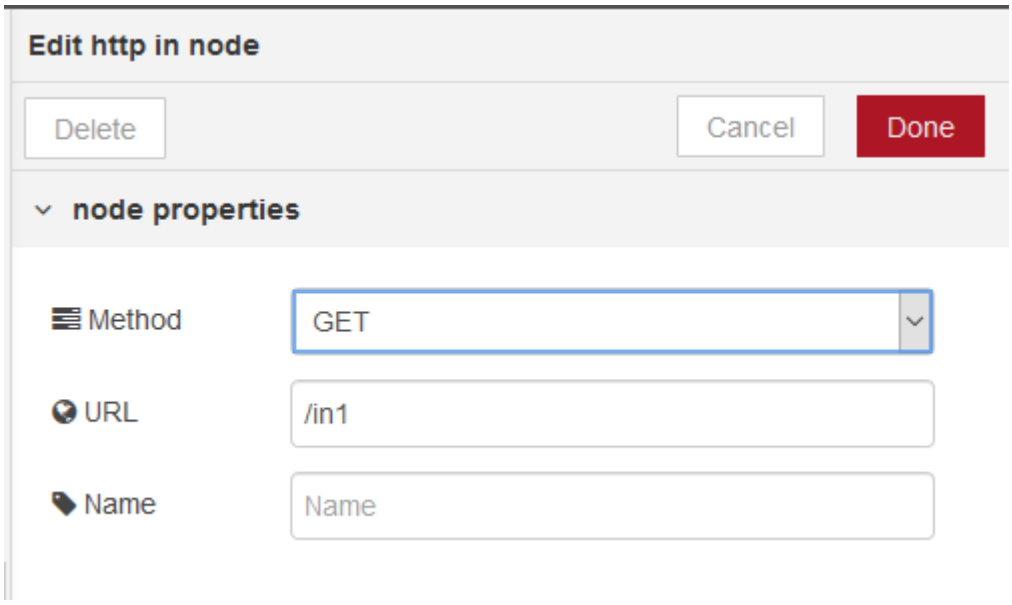
Http in

Створює точку для з`єднання HTTP для створення веб-служб. На виході формує:

- Payload (object) - Для запиту GET містить об'єкт з параметрами рядка запиту. В іншому випадку, містить тіло запиту HTTP.
- req (object) - Об'єкт запиту HTTP. Цей об'єкт містить кілька властивостей, які надають інформацію про запит.
    - _readableState –
    - readable
    - domain
    - _events
    - _eventsCount –
    - socket –
    - connection –
    - httpVersionMajor –
    - httpVersionMinor –
    - httpVersion –
    - complete –
    - `headers` – об’єкт, що вміщує заголовок запиту HTTP .

- rawHeaders –
- trailers –
- rawTrailers –
- aborted
- upgrade
- url –
- method –
- statusCode –
- statusMessage –
- client –
- _consuming –
- _dumped –
- next –
- baseUrl –
- originalUrl –
- _parsedUrl –
- `params` – об’єкт, що вміщує будь-які маршрутні параметри.
- `query` – об’єкт, що вміщує любі строкові параметри запиту .
- res -
- `body` – тіло вхідного запиту. Формат залежить від запиту.
- _passport –
- _parsedOriginalUrl –
- route -
- `cookies` – об’єкт, що вміщує cookies для запиту.
- signedCookies -
- `files` - якщо активовано у вузлі, об’єкт містить будь-які файли завантажені з запитом POST.

- res (object) - HTTP об’єкт відповіді. Ця властивість не повинна використовуватися безпосередньо; ознайомтеся з документами на вузол `HTTP Response` для формування правильної відповіді на запит. Це властивість має залишатися прикріпленим до повідомлення, переданого вузлу відповіді.

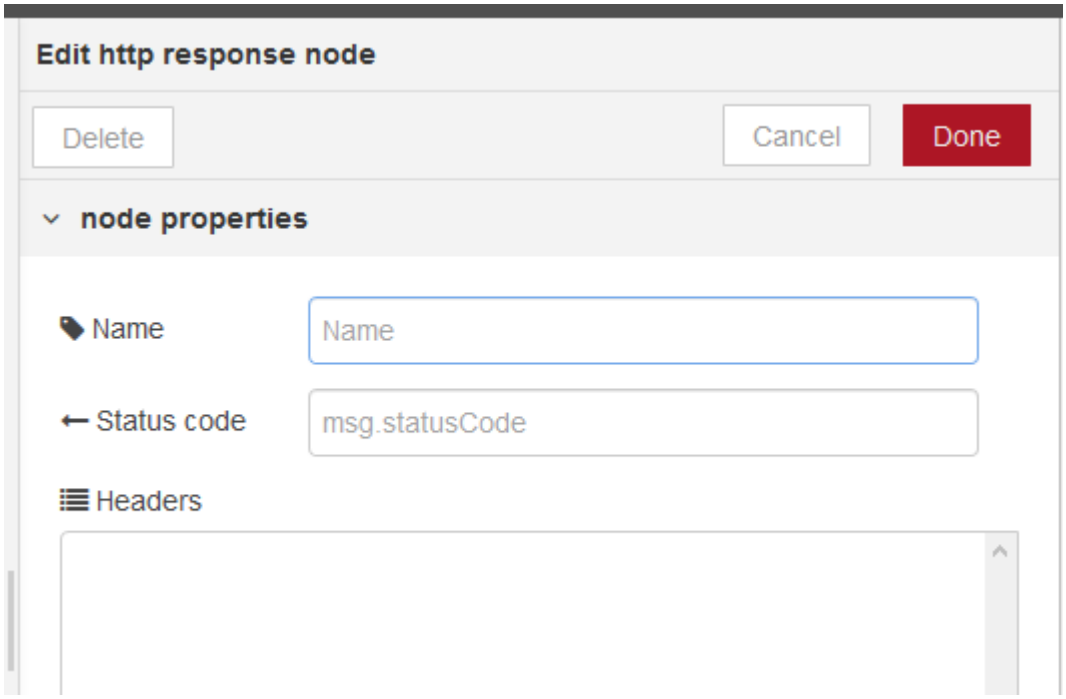


Вузол прослуховує конфігурований шлях для запитів певного типу. Шлях може бути повністю означений, наприклад `/user`, або включати іменовані параметри, які приймають будь-яке

значення, наприклад `/user/:name`. Коли використовуються іменовані параметри, їх фактичне значення в запиті може бути доступне за посиланнями `msg.req.params`.
Для запитів, які включають тіло, наприклад POST або PUT, вміст запиту доступний як `msg.payload`.
Якщо тип вмісту запиту може бути визначений, тіло буде проаналізовано до будь-якого відповідного типу. Наприклад, `application/json` буде парсений до його представлення в об'єкти JavaScript.
Примітка: цей вузол не надсилає відповіді на запит. Потік повинен містити вузол HTTP Response для завершення запиту.

Http response

Надсилає відповіді на запити, отримані від вузла HTTP Input.



В якості вхідного значення приймає наступні властивості повідомлень:

- payload (string) – тіло відповіді
- statusCode (number) – якщо встановлений,використовується в якості статусного коду відповіді. За замовченням 200
- headers (object) – заголовки, якщо встановлений забезпечує HTTP заголовки, які включаються у відповідь
- cookies (object) – якщо встановлений, може бути використаний для встановлення або видалення куків (cookies)

`statusCode` і `headers` також можуть бути встановлені в налаштуваннях самого вузла. У цьому випадку їх не можна перевизначити відповідними властивостями повідомлення.

Властивість `cookies` повинна бути об'єктом пар імен/значень. Значення може бути або рядком для встановлення значення куки з параметрами за замовчуванням, або це може бути об'єктом опцій.

Наступний приклад встановлює два файли cookies - один з них називається `name` зі значенням `nick`, інший називається `session` зі значенням `1234` з терміном дії 15 хвилин.

```
msg.cookies = {
    name: 'nick',
    session: {
        value: '1234',
        maxAge: 900000
    }
}
```

Допустимі опції:

- `domain` - (String) ім’я домену для куки
- `expires` - (Date) expiry date in GMT. If not specified or set to 0, creates a session cookie
- `maxAge` - (String) термін дії відносно поточного часу в мілісекундах
- `path` - (String) шлях куки, за замовченням «/»
- `value` - (String) значення для куки

Для видалення куки встановлюється в `null`

WebSocket

Налаштування клієнтських і серверних з’єднань

Для налаштування серверного ресурсу WebSocket для Node-RED використовуються конфігураційні вузли WebSocket-listener. Вхідний порт для з’єднання буде тим самим, що і порт для конфігурування/WEB, тобто типово, 1880. Тому в конфігураційному вузлі вказується тільки частина URI що вказує розміщення ресурсу на сервері.

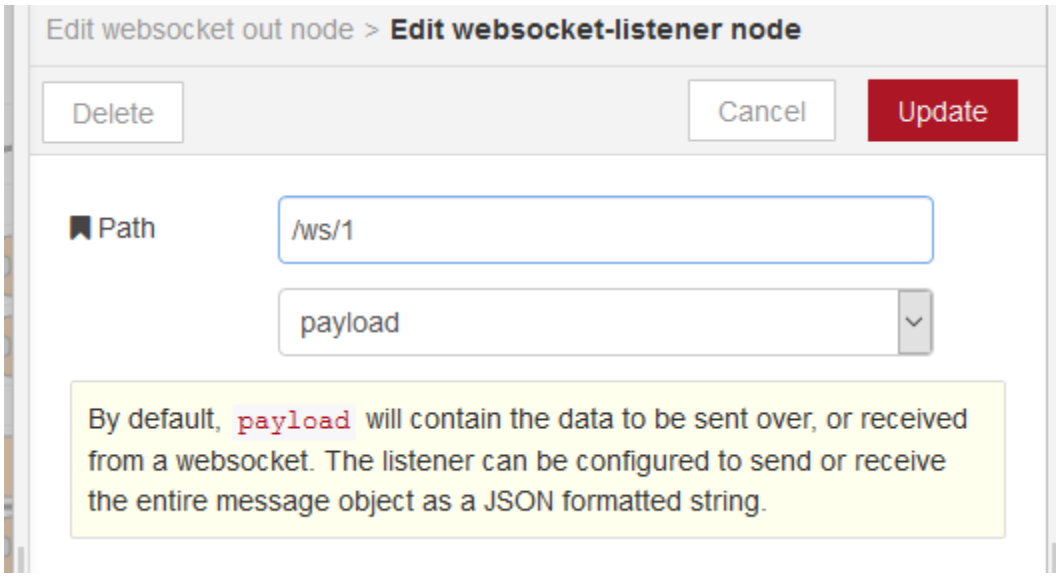


рис.1.Конфігураційний вузол WebSocket-listener

Для налаштування з’єднання клієнтського вузлу WebSocket з Node-RED використовуються конфігураційні вузли WebSocket-client. Для означення схеми на початку URL вказується ws:// для звичайного або wss:// для захищеного з’єднання. Далі вказується повний шлях до ресурсу, включаючи домен (або IP), порт (якщо не 80) та інша частина URL.

.

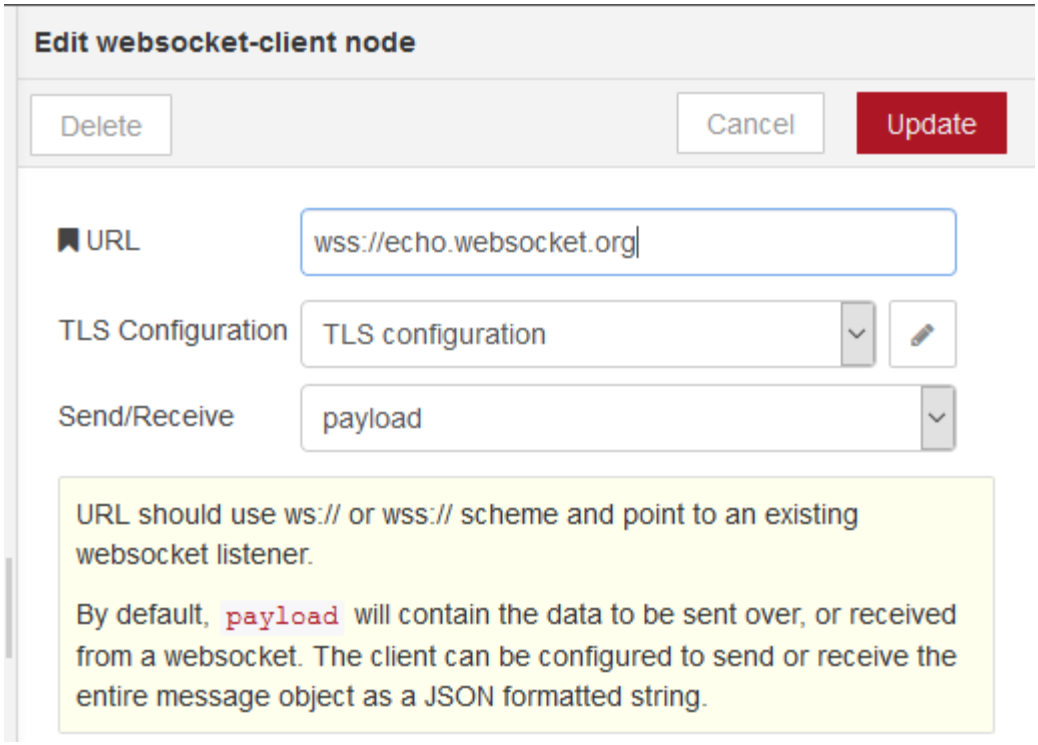
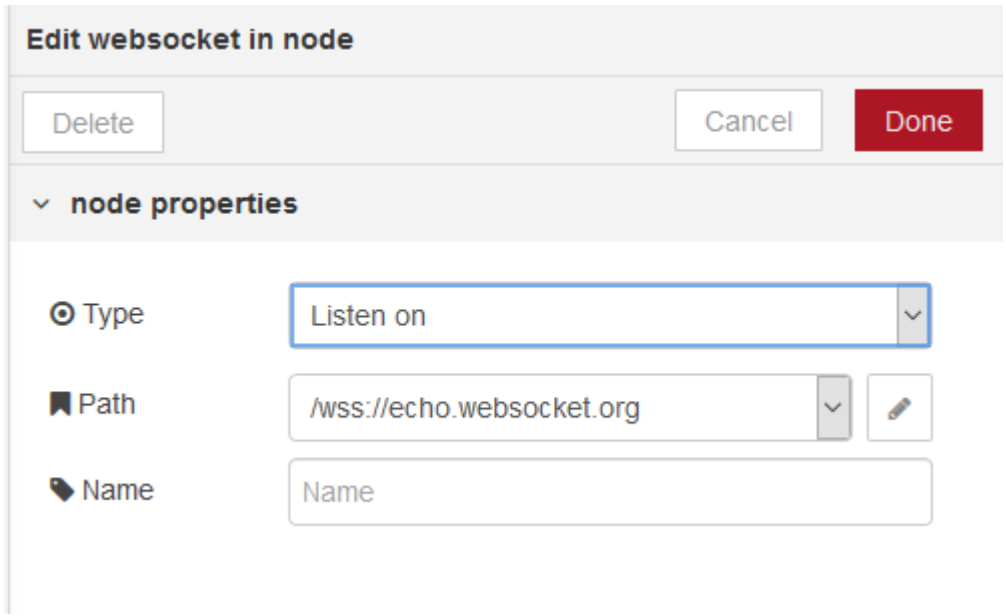


рис.2.Конфігураційний вузол WebSocket- client

WebSocket in

Вхідний вузол WebSocket.



За замовчуванням, дані, отримані з WebSocket, будуть знаходитись у `msg.payload`. Сокет може бути налаштований так, щоб очікувати правильно сформованого JSON-рядку, і в цьому випадку він розбирає JSON і надсилає отриманий об'єкт як ціле повідомлення.

https://uk.wikipedia.org/wiki/WebSocket

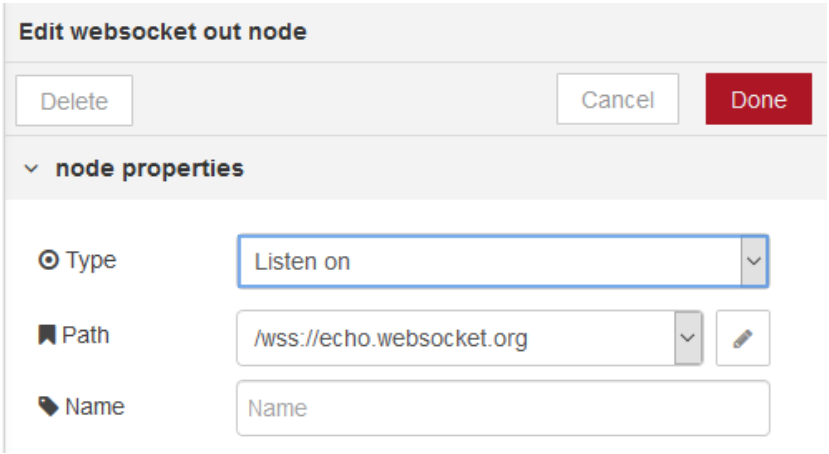
https://tproger.ru/translations/what-are-web-sockets/

https://www.websocket.org/echo.html

https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/

WebSocket out

Вихідний вузол WebSocket.



За замовчуванням `msg.payload` відправляє через WebSocket. Сокет може бути зконфігурований так, щоб кодувати весь об'єкт `msg` як рядок JSON і посилати його через WebSocket.

Якщо повідомлення, що надходить на цей вузол, запущено в вузлі WebSocket In, повідомлення буде відправлено клієнту, який ініціював потік. В іншому випадку повідомлення буде транслюватися всім підключеним клієнтам.

Якщо ви хочете передати повідомлення, яке розпочалося в вузлі WebSocket In, ви повинні видалити властивість `msg._session` з потоку.

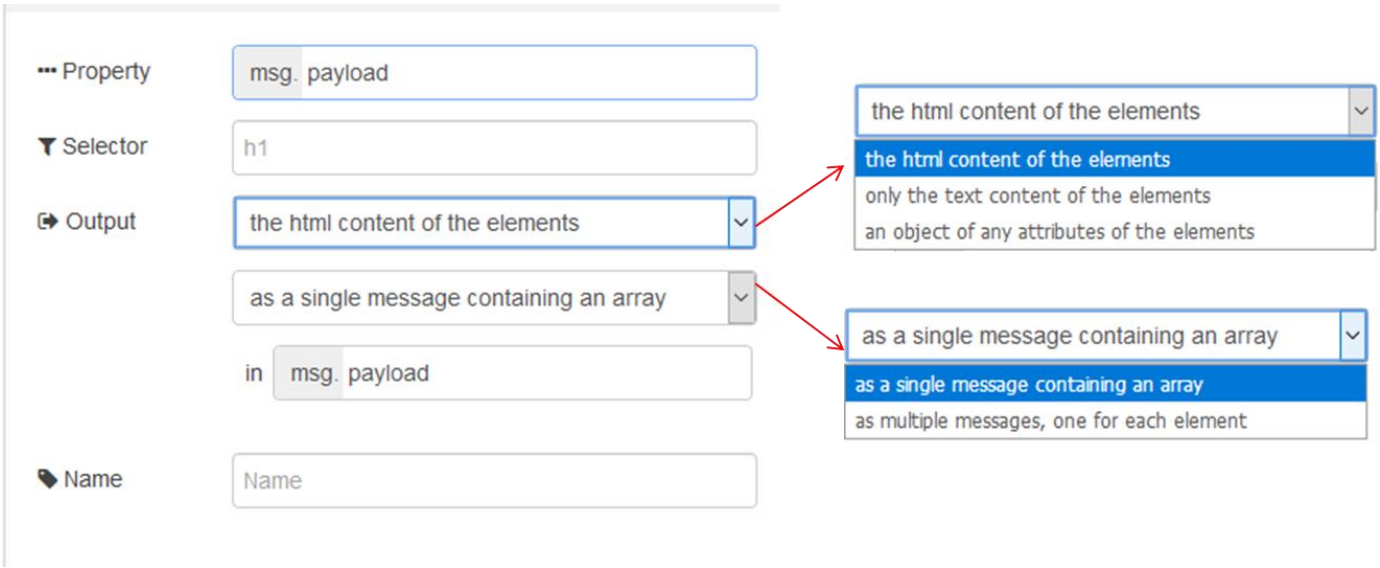
PARSING

HTML

Витягує елементи з HTML-документа, що міститься в `msg.payload` за допомогою селекторів CSS.

В якості вхідного значення приймає наступні властивості повідомлень:

- payload (string) – html- рядок з якого вилучаються елементи.
- select (string) - селектор, може бути використане це значення властивості msg, якщо воно не налаштовано на панелі редагування.



На виході формує:

- payload (array | string) - результатом може бути одне повідомлення з корисним навантаженням, що містить масив відповідних елементів, або кілька повідомлень, кожен з яких містить відповідний елемент. Якщо надсилаються декілька повідомлень, вони також мають набір `parts`.

Цей вузол підтримує комбінацію селекторів CSS і jQuery. Докладніше про підтримуваний синтаксис див. тут або Документацію css-select.

JSONata

JSONata - це мова запитів і перетворень для даних JSON. Для маніпулювання та об'єднання відібраних даних передбачено набір вбудованих операторів і функцій, а результати запитів можуть бути відформатовані в будь-яку вихідну структуру JSON, використовуючи звичний JSON-об'єкт і синтаксис масиву. У поєднанні з можливістю створення функцій, визначених користувачем, можна розробити додаткові вирази для вирішення будь-яких JSON-запитів і завдань трансформації.
Документація - http://docs.jsonata.org
Перевірка на прикладі - http://try.jsonata.org/

Приклад використання в застосунках Node-RED

JSONata використовується в багатьох вузлах Node-RED. На рис.1 показаний приклад використання в вузлі *change*.

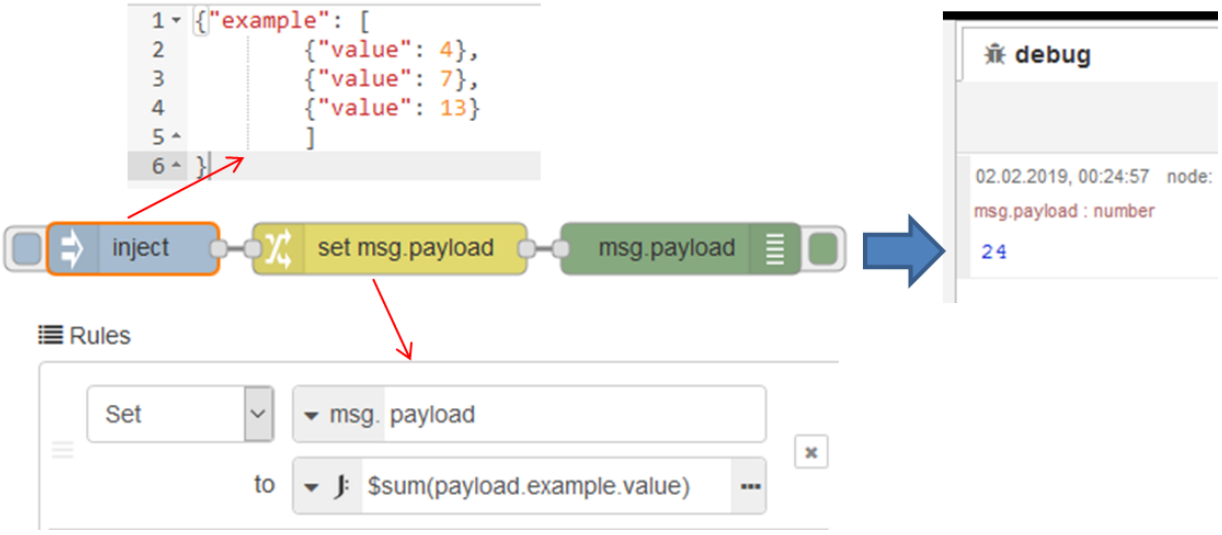


рис.1.

Прості запити

Для підтримки вилучення значень зі структури JSON означений **синтаксис шляху розташування**. Це дозволяє вибрати всі можливі значення в документі, які відповідають вказаному **шляху розташування** (location path). В JSON є дві структурні конструкції - це об'єкти і масиви. Нижче показаний синтаксис роботи з ними в JSONata.

Прості приклади навігації об'єктами JSON

У таблиці 1 показні прості приклади задавання пошуку в JSONata. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

При вказівки простих запитів шляху слід дотримуватися наступних правил. При пошуку полів з пробілами вони вказуються в спеціальних лапках ` `. Якщо індекс в шуканому масиві не є цілим, він округляється до цілого. Негативне значення індексу в масиві шукає значення відраховане з кінця, наприклад `arr[-1]` бере останній елемент, а `arr[-2]` – бере передостанній. Якщо індекс не вказується – береться весь масив. Якщо при цьому масив містить об'єкти, а шлях розташування вибирає поля в цих об'єктах, то для кожного об'єкта в масиві буде запрошено вибір.

Таблиця 1.

Приклад	JSONata	Result

<pre>{
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Address": {
  "Street": "Hursley Park",
  "City": "Winchester",
  "Postcode": "SO21 2JN"
  },
  "Phone": [
  {
  "type": "home",
  "number": "0203 544 1234"
  },
  {
  "type": "office",
  "number": "01962 001234"
  },
  {
  "type": "office",
  "number": "01962 001235"
  },
  {
  "type": "mobile",
  "number": "077 7700 1234"
  }
  ],
  "Email": [
  {
  "type": "work",
  "address": ["fred.smith@my-work.com", "fsmith@my-work.com"]
  },
  {
  "type": "home",
  "address": ["freddy@my-social.com", "frederic.smith@very-serious.com"]
  }
  ],
  "Other": {
  "Over 18 ?": true,
  "Misc": null,
  "Alternative.Address": {
  "Street": "Brick Lane",
  "City": "London",
  "Postcode": "E1 6RF"
  }
  }
}</pre>
	<pre>Surname</pre>	<pre>"Smith"</pre>

	<pre>Age</pre>	<pre>28</pre>
	<pre>Address.City</pre>	<pre>"Winchester"</pre>
	<pre>Other.Misc</pre>	<pre>null</pre>
	<pre>Other.Nothing</pre>	*undefined*
	<pre>Other.`Over 18 ?`</pre>	<pre>true</pre>
	<pre>Phone[0]</pre>	<pre>{ "type": "home", "number": "0203 544 1234" }</pre>
	<pre>Phone[-1]</pre>	<pre>{ "type": "mobile", "number": "077 7700 1234" }</pre>
	<pre>Phone[-2]</pre>	<pre>{ "type": "office", "number": "01962 001235" }</pre>
	<pre>Phone[8]</pre>	*undefined*
	<pre>Phone[0].number</pre>	<pre>"0203 544 1234"</pre>
	<pre>Phone.number</pre>	<pre>["0203 544 1234", "01962 001234", "01962 001235", "077 7700 1234"]</pre>
	<pre>Phone.number[0]</pre>	<pre>["0203 544 1234", "01962 001234", "01962 001235", "077 7700 1234"]</pre>
	<pre>(Phone.number)[0]</pre>	<pre>"0203 544 1234"</pre>

	<pre>Phone[[0..1]]</pre>	<pre>[
		  { "type": "home", "number": "0203 544 1234" },
		  { "type": "office", "number": "01962 001234" }
		]</pre>

Пусті послідовності та послідовності з одним елементом

Візьмемо приклад:

```
[
  { "ref": [ 1,2 ] },
  { "ref": [ 3,4 ] }
]
```

На верхньому рівні ми маємо масив, а не об'єкт. Якщо ми хочемо вибрати перший (0-й) об'єкт у цьому масиві, то необхідно вказати назву об’єкту верхнього рівня, до якого треба додати [0]. Ми не можемо використовувати [0] самостійно, тому що стикаємося з синтаксисом конструктора масиву. Однак, ми можемо використовувати посилання контексту *$* (*context*) для посилання на початок документа наступним чином:

Приклад	JSONata	Result	Коментар
[{ "ref": [1,2] }, { "ref": [3,4] }]	$[0]	{ "ref": [1,2] }	$ на початку виразу відноситься до всього вхідного документа
	$[0].ref	[1,2]	.ref тут повертає весь внутрішній масив
	$[0].ref[0]	1	повертає елемент на першу позицію внутрішнього масиву
	$.ref	[1, 2, 3, 4]	Незважаючи на структуру вкладеного масиву, результуючий вибір сплющується в один плоский масив. Початкова вкладена структура вхідних масивів втрачається.

Предикативні запити (Predicate Queries)

Предикати

На будь-якому кроці шляху розташування виділені елементи можуть бути відфільтровані за допомогою **предиката** - `[expr]`, де `expr` аналізується як вираз, що повертає булеве значення. Кожен елемент у виділенні перевіряється як вираз, якщо він дорівнює `true`, то елемент залишається у виділенні, якщо `false`, він видаляється з виділення. Вираз оцінюється відносно поточного (контекстного) елемента, що перевіряється, тому, якщо вираз предиката виконує навігацію, то вона проводиться відносно цього контексту.

У таблиці 2 показні приклади предикатів. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 2.

Приклад	JSONata	Result	Коментар
{"Phone": [{"type": "home", "number": "0203 544 1234" }, {"type": "office", "number": "01962 001234" }, {"type": "office", "number": "01962 001235"}, {"type": "mobile", "number": "077 7700 1234" }]}	Phone[type='mobile']	{"type": "mobile", "number": "077 7700 1234" }	Вибирає елементи Phone, у яких є поле типу "mobile".
	Phone[type='mobile'].number	"077 7700 1234"	вибір номерів мобільних телефонів
	Phone[type='office'].number	["01962 001234", "01962 001235"]	вибір офісних мобільних телефонів

Одноелементний масив і еквівалентність значень

У виразі або підвиразі JSONata будь-яке значення (яке не є масивом) і масив, що містить тільки це значення, вважаються еквівалентними. Це дозволяє складати композицію таким чином, щоб шляхи розташування, які витягують одне значення з об'єктів і шляхи розташування, які витягують декілька значень з масивів, можуть бути використані як вхідні дані для інших виразів без використання іншого синтаксису для двох форм.

Наприклад (див вихідний текс JSON вище):

`Address.City` повертає `"Winchester"`

`Phone[0].number` повертає `"0203 544 1234"`

`Phone[type='home'].number` повертає `"0203 544 1234"`

`Phone[type='office'].number` повертає `[ "01962 001234", "01962 001235" ]`

При наступній обробці поверненого з вираження JSONata значення, може бути бажаним мати результати в узгодженому форматі, незалежно від того, скільки співпадінь було і відповідно який формат поверненого значення (масив або одне значення). У перших двох виразах вище, очевидно, що кожен вираз звертається до одного значення в структурі і має сенс повернути саме це значення. В останніх двох виразах, однак, не є очевидним, скільки значень буде повернено, тому при наступній обробці прийдеться враховувати різні формати, що не є зручним.

Для таких випадків вираз можна змінити таким чином, щоб повернути масив, навіть якщо збігається лише одне значення. Це робиться шляхом додавання порожніх квадратних дужок *[]* до кроку в шляху розташування. Наведені вище приклади можуть бути переписані так, щоб завжди повертати масив, це виглядає наступним чином:

- `Address[].City` повертає `[ "Winchester"]`
- `Phone[0][].number` повертає `[ "0203 544 1234" ]`
- `Phone[][type='home'].number` повертає `[ "0203 544 1234" ]`
- `Phone[type='office'].number[]` повертає `[ "01962 001234", "01962 001235" ]`

Зауважимо, що *[]* можна розмістити у будь-якому місці предикатів і на будь-якому кроці вираження шляху

Шаблони заміни

Використовуйте * замість імені поля для вибору всіх полів в об'єкті. У таблиці 3 показні приклади предикату *. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 3.

Приклад	JSONata	Result
`{ "FirstName": "Fred", "Surname": "Smith", "Age": 28, "Address": { "Street": "Hursley Park", "City": "Winchester", "Postcode": "SO21 2JN" } }`	`Address.*`	`[ "Hursley Park", "Winchester", "SO21 2JN" ]`
	`*.Postcode`	`"SO21 2JN"`

Шаблон нащадка ** замість * буде відбирати всіх нащадків (багаторівневий шаблон).

Для прикладу з таблиці 1

`**.Postcode` поверне `[ "SO21 2JN", "E1 6RF" ]`

Функції та вирази

Строкові вирази

У виразах шляху можна використовувати літеральні константи, помістивши символи в лапки: подвійні " або одинарні ‘ (як і рядки JSON). Рядки можна комбінувати за допомогою оператора конкатенації &, цей оператор поєднує два рядки, які повертаються виразами. Це єдиний оператор, який намагатиметься збирати операнди разом до очікуваного рядкового типу.

У таблиці 4 показні приклади оператору &. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 4.

Приклад	JSONata	Result
`{ "FirstName": "Fred", "Surname": "Smith", "Age": 28, "Address": { "Street": "Hursley Park", "City": "Winchester", "Postcode": "SO21 2JN" } }`	`FirstName & ' ' & Surname`	`"Fred Smith"`
	`Address.(Street & ', ' & City)`	`"Hursley Park, Winchester"`
	`5&0&true`	`"50true"`

Числові вирази

Вирази шляхів, які вказують на числове значення, повернуть це значення як числовий тип. JSONata також може використовувати числові контстанти за тими ж правилами, що і для JSON-чисел. Числові літетерали та вирази можуть бути використані в розрахунках результатів з використанням звичайних математичних операторів. Підтримувані оператори:

- `+` додавання
- `-` віднімання
- `*` множення
- `/` ділення
- `%` остача від ділення

У таблиці 5 показні приклади арифметичних операторів. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (скопіюйте приклад в поле Exerciser).

Таблиця 5.

Приклад	JSONata	Result	Коментар
`{ "Numbers": [1, 2.4, 3.5, 10, 20.9, 30] }`	`Numbers[0] + Numbers[1]`	`3.4`	`додати 2 числа`
	`Numbers[0] - Numbers[4]`	`-19.9`	`віднімання`
	`Numbers[0] * Numbers[5]`	`30`	`множення`
	`Numbers[0] / Numbers[4]`	0.04784688995215	ділення
	`Numbers[2] % Numbers[5]`	3.5	остача від ділення

Вирази порівняння

В предикатах можна використоувати оператори порівнянн двох значень, кі повертають логічні значення *true* або *false*. Підтримувані оператори:

- = дорівнює
- != не дорівнює
- < менше ніж
- <= менше ніж чи дорівнює
- > більше ніж
- >= більше або дорівнює ніж
- `in` значення міститься в масиві

У таблиці 6 показні приклади виразів порівнянь. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (скопіюйте приклад в поле Exerciser).

Таблиця 6.

Приклад	JSONata	Result
`{ "Numbers": [1, 2.4, 3.5, 10, 20.9, 30] }`	`Numbers[0] = Numbers[5]`	*false*
	`Numbers[0] != Numbers[4]`	*true*
	`Numbers[1] < Numbers[5]`	*true*
	`Numbers[1] <= Numbers[5]`	*true*
	`Numbers[2] > Numbers[4]`	*false*
	`Numbers[2] >= Numbers[4]`	*false*

Булеві вирази

Використовується для об'єднання булевих результатів для підтримки більш складних предикатних виразів. Підтримувані оператори:

- `and`
- `or`

Зверніть увагу, що `not` підтримується як функція, а не оператор.

У таблиці 7 показні приклади булевих виразів. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (скопіюйте приклад в поле Exerciser).

Таблиця 7.

Приклад	JSONata	Result
`{"Numbers": [1, 2.4, 3.5, 10, 20.9, 30] }`	`(Numbers[2] != 0) and (Numbers[5] != Numbers[1])`	*false*
	`(Numbers[2] != 0) or (Numbers[5] = Numbers[1])`	*true*

Структурування результату

Тут розглядається способи представлення результатів на виході JSONata.

Конструктор масиву

Як зазначалося раніше, якщо шлях розташування у вхідному документі відповідає декільком значенням, ці значення повертаються як масив. Значення у документі можуть бути як

об'єктами так і масивами, але значення що повертаються знаходяться на верхньому рівні у вигляді масиву.

Можна побудувати додаткову структуру в результуючому масиві, вказавши у виразі шляху розташування конструктор масивів (або об'єктів). У будь-якій точці шляху розташування, де очікується посилання на поле, можна вставити пару квадратних дужок [], щоб вказати, що результати вираження в цих дужках повинні міститися в новому масиві на виході. Коми використовуються для розділення декількох виразів у конструкторі масиву.

У таблиці 8 показні приклади конструкторів масивів. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 8.

Приклад	JSONata	Result	Коментар

<pre>{
  "FirstName": "Fred",
  "Surname": "Smith",
  "Age": 28,
  "Address": {
  "Street": "Hursley Park",
  "City": "Winchester",
  "Postcode": "SO21 2JN"
  },
  "Email": [
  {
  "type": "work",
  "address": ["fred.smith@my-work.com", "fsmith@my-work.com"]
  },
  {
  "type": "home",
  "address": ["freddy@my-social.com", "frederic.smith@very-serious.com"]
  }
  ],
  "Other": {
  "Over 18 ?": true,
  "Misc": null,
  "Alternative.Address": {
  "Street": "Brick Lane",
  "City": "London",
  "Postcode": "E1 6RF"
  }
  }
}</pre>
	<pre>Email.address</pre>	<pre>[
		  "fred.smith@my-work.com",
		  "fsmith@my-work.com",
		  "freddy@my-social.com",
		"frederic.smith@very-serious.com"
		]</pre>
			Чотири адреси електронної пошти повертаються в плоскому масиві

	<pre>Email.[address]</pre>	<pre>[
		["fred.smith@my-work.com",
		"fsmith@my-work.com"],
		["freddy@my-social.com",
		"frederic.smith@very-serious.com"]
		]</pre>
			Кожен об'єкт електронної пошти генерує масив адрес

	<pre>[Address, Other.`Alternative.Address`].City</pre>	<pre>["Winchester", "London"]</pre>	Вибирає значення `City` як з об'єкту `Address` так і з `Alternative.Address`.

Конструктор об'єкту

Подібно до того, як можна побудувати масиви, так само можуть бути побудовані на виході об'єкти JSON. У будь-якій точці шляху розташування, де очікується посилання на поле, можна використати пару фігурних дужок {}, що містять пари ключ/значення, розділені комами, з кожним ключем і значенням, розділеними двокрапкою: {key1: value2, key2: value2}. Ключі та значення можуть бути літералами або можуть бути виразами. Ключ повинен бути або рядком, або виразом, який оцінюється до рядка.

Коли за виразом, що вибирає кілька значень, йде конструктор об'єктів, той створить єдиний об'єкт з парою ключ/значення для кожного з цих значень контексту. Якщо потрібний

масив об'єктів (один для кожного значення контексту), то конструктор об'єкта повинен слідувати за точкою '.' .

У таблиці 9 показні приклади конструкторів об’єктів. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 9.

Приклад	JSONata	Result	Коментар
`{"Phone": [ {"type": "home", "number": "0203 544 1234" }, {"type": "office", "number": "01962 001234" }, {"type": "office", "number": "01962 001235"}, {"type": "mobile", "number": "077 7700 1234" }]}`	`Phone.{type: number}`	`[ { "home": "0203 544 1234" }, { "office": "01962 001234" }, { "office": "01962 001235" }, { "mobile": "077 7700 1234" } ]`	Створює масив об'єктів (по одному для кожного телефону).
	`Phone{type: number}`	`{ "home": "0203 544 1234", "office": [ "01962 001234", "01962 001235" ], "mobile": "077 7700 1234" }`	Об'єднує пари ключ/значення в один об'єкт. Докладніше див. Grouping using object key expression
	`Phone{type: number[]}`	`{ "home": [ "0203 544 1234" ], "office": [ "01962 001234", "01962 001235" ], "mobile": [ "077 7700 1234" ] }`	Об'єднує пари ключ/значення в один об'єкт. У цьому випадку для узгодженості всі числа групуються в масиви. Див. Singleton array and value equivalence.

Літерали JSON

Конструктори масиву та об'єкта використовують стандартний синтаксис JSON для масивів і об'єктів. На додаток до цих значень можна ввести в вираз інші типи даних JSON, використовуючи власний синтаксис JSON:

- strings - `"hello world"`
- numbers - `34.5`
- Booleans - `true` or `false`
- nulls - `null`
- objects - `{"key1": "value1", "key2": "value2"}`
- arrays - `["value1", "value2"]`

JSONata є надмножиною JSON. Це значить, що будь-який дійсний документ JSON також є дійсним виразом JSONata. Це властивість дозволяє використовувати документ JSON як шаблон для бажаного виводу, а потім замінити його частинами виразами для вставки даних у вихідні дані з вхідного документа.

Запити композиції (Query composition)

У JSONata все є виразами (*expression*). Вираз містить значення (*values*), функції (*functions*) та оператори (*operators*), які при оцінюванні (*evaluated*) виробляють результуюче значення. Функції та оператори застосовуються до тих значень, які в свою чергу самі можуть бути результатами оцінювання під-виразів. Таким чином, мова передбачає складені (ієрархічні) вирази. Приклади виразів:

- `(5 + 3) * 4`
- `Product.(Price * Quantity)`

Використовуйте ‘блок коду’ - декілька виразів, розділених крапкою з комою `(expr1; expr2; expr3)`. Кожен вираз в блоці оцінюється в послідовності відповідно до порядку, з блоку повертається результат останнього виразу.

Сортування, групування і агрегація

Сортування

Масиви містять впорядкований набір значень. Якщо потрібно змінити порядок значень, то масив необхідно відсортувати. У JSONata існує два способи сортування масиву:

1. Використовуючи функцію `$sort()`.
2. Використовуючи оператор order-by (^(...)).

Оператор order-by є зручним синтаксисом, який може використовуватися безпосередньо у виразі шляху для сортування послідовностей результатів у порядку зростання або спадання. Функція `$sort()` вимагає більше синтаксису для запису, але є більш гнучкою і підтримує функції користувацьких компараторів.

У таблиці 10 показні приклади використання функції і оператору сортування. Можна перевірити ці приклади, перейшовши на сайт http://try.jsonata.org/ (виберіть приклад Address).

Таблиця 10.

Приклад	JSONata	Result	Коментар
`{ "Phone": [ { "type": "home", "number": "0203 544 1234" }, { "type": "office", "number": "01962 001234" }, { "type": "office", "number": "01962 001235" }, { "type": "mobile", "number": "077 7700 1234" } ] }`	`Phone^(>number)`	`[ { "type": "mobile", "number": "077 7700 1234" }, { "type": "home", "number": "0203 544 1234" }, { "type": "office", "number": "01962 001235" }, { "type": "office", "number": "01962 001234" } ]`	`В порядку спадання поля number`
	`$sort(Phone, function($l, $r) {`	`[ { "type": "office", "number": "01962`	В порядку зростання 3-х перших символів

	`$substring($l.number,1,3) > $substring($r.number,1,3))`	`001234" }, { "type": "office", "number": "01962 001235" }, { "type": "home", "number": "0203 544 1234" }, { "type": "mobile", "number": "077 7700 1234" } ]`	поля `number`

Групування

Синтаксис конструктора об'єктів JSONata дозволяє вказувати вираз для означення ключа в будь-якій парі ключ/значення (значення, очевидно, також може бути виразом). Вираз ключа повинен бути рядком. Результат кожної пари виразів ключ/значення вставляється в результуючий об'єкт JSON. Якщо оцінка будь-якого виразу для ключа призводить до того, що ключ вже є в об'єкті результату, то результат цього виразу буде згруповано з значенням, яке вже пов'язано з цим ключем. Зауважте, що значення виразу не оцінюються, доки не буде виконано все групування, це дозволяє оцінювати вирази агрегації по колекції елементів для кожної групи.

Приклад	JSONata	Result
`див. http://try.jsonata.org/ (приклад Invoice)	`Account.Order.Product{`Product Name`: Price}`	`{ "Bowler Hat": [ 34.45, 34.45 ], "Trilby hat": 21.67, "Cloak": 107.99 }`
	`Account.Order.Product { `Product Name`: {"Price": Price, "Qty": Quantity} }`	`{ "Bowler Hat": { "Price": [ 34.45, 34.45 ], "Qty": [ 2, 4 ] }, "Trilby hat": { "Price": 21.67, "Qty": 1 }, "Cloak": { "Price": 107.99, "Qty": 1 } }`

Зверніть увагу, що у наведеному вище прикладі вираз значення об'єднав усі ціни разом і всі величини разом у окремі масиви. Це пояснюється тим, що значення контексту - це послідовність всіх згрупованих продуктів, а вираз `Price` вибирає всі ціни з усіх продуктів. Якщо ви хочете зібрати ціну і кількість в окремі об'єкти, то вам необхідно оцінити конструктор об'єкта *for each* продукту в послідовності контексту. Наступний приклад показує це.

Приклад	JSONata	Result	Примітки
`див.	`Account.Order.Pro	{	Явне використання

http://try.jsonata.org/ (приклад Invoice)	duct {
	`Product Name`: $.{"Price": Price, "Qty": Quantity}
	}
		"Bowler Hat": [
		{ "Price": 34.45, "Qty": 2 },
		{ "Price": 34.45, "Qty": 4 }
		],
		"Trilby hat": { "Price": 21.67, "Qty": 1 },
		"Cloak": { "Price": 107.99, "Qty": 1 }
		}
			$.{ ... } для створення об'єкта для кожного елементу в групі

	Account.Order.Product{`Product Name`: $.(Price*Quantity)}	{
		"Bowler Hat": [68.9, 137.8],
		"Trilby hat": 21.67,
		"Cloak": 107.99
		}
			Множить ціну на кількість для кожного продукту в кожній групі

Агрегація

Часто запити просто потрібні для повернення агрегованих результатів з набору відповідних значень. Доступно ряд функцій агрегації, які повертають єдине агреговане значення при застосуванні до масиву значень.

Приклад	JSONata	Result	Примітки
`див. http://try.jsonata.org/ (приклад Invoice)	$sum(Account.Order.Product.Price)	198.56	
	$sum(Account.Order.Product.(Price*Quantity))	336.36	

Числові функції агрегації доступні за посиланням.

Програмні конструкції

До цих пір ми ввели всі частини мови, які дозволяють нам витягувати дані з вхідного JSON-документа, об'єднувати дані за допомогою рядкових і числових операторів і форматувати структуру вихідного документа JSON. Тим не менше JSONata можна використовувати як повноцінну мову програмування Тьюрінга.

Коментарі

У виразах JSONata можуть бути використані коментарі з використанням синтаксису стилю мови «C».

```
/* Long-winded expressions might need some explanation */
(
  $pi := 3.141592653589793238462643;
  /* JSONata is not known for its graphics support! */
  $plot := function($x) {(
    $floor := $string ~> $substringBefore(?, '.') ~> $number;
    $index := $floor(($x + 1) * 20 + 0.5);
    $join([0..$index].('.')) & 'O' & $join([$index..40].('.'))
  )};

  /* Factorial is the product of the integers 1..n */
  $product := function($a, $b) { $a * $b };
  $factorial := function($n) { $n = 0 ? 1 : $reduce([1..$n], $product) };

  $sin := function($x){ /* define sine in terms of cosine */
    $cos($x - $pi/2)
  };
  $cos := function($x){ /* Derive cosine by expanding Maclaurin series */
    $x > $pi ? $cos($x - 2 * $pi) : $x < -$pi ? $cos($x + 2 * $pi) :
      $sum([0..12].($power(-1, $) * $power($x, 2*$) / $factorial(2*$)))
  };

  [0..24].$sin($*$pi/12).$plot($)
)
```

Подивіться на цей приклад в дії за цим посиланням.

Побудова умов

Умовні конструкції «Якщо/тоді/інакше» будуються з використанням умовних операторів "? :"

```
predicate ? expr1 : expr2
```

Оцінюється вираз `predicate`. Якщо вираз повертає `true` тоді оцінюється вираз `expr1` і повертається його значення, інакше обробляється вираз `expr2`.

Приклад	JSONata	Result	Примітки
див. http://try.jsonata.org/	`35.6>20.1 ? 12 : 10`	12	
	`35.6<20.1 ? 12 : 10`	10	

Змінні

Будь які назви, що починаються з знаку '$' є змінними. **Змінна** – це поіменоване посилання на значення. Значення може бути одним із будь-яких типів серед системних типів. Є також вбудовані в JSONata змінні:

- `$` - змінна без імені посилається на значення контексту у будь якій точці вхідної ієрархії JSON.

- `$$` - корінь входу JSON. Тільки потребується у випадках для виходу з теперішнього контексту для тимчасового переходу вниз в інший шлях. Наприклад для перехресного посилання або об'єднання даних.
- Рідні (вбудовані) функції. Див функції.

Зв'язування змінних

Значення можуть бути зв'язані зі змінними наступним чином:

```
$var_name := "value"
```

Збережене значення може бути позначено пізніше за допомогою виразу `$var_name`

Область видимості змінних обмежена 'блоком' в якому відбувалось зв'язування. Наприклад:

```
Invoice.(
  $p := Product.Price;
  $q := Product.Quantity;
  $p * $q
)
```

`Повертає значення Price` помножений на `Quantity в Product з Invoice.`

Функції

Функції є першокласним типом і можуть бути збережені в змінних подібно іншим типам. JSONata забезпечує глобальну бібліотеку вбудованих функцій, що назначені змінним в глобальній області видимості. Наприклад, `$uppercase` вміщує функцію, яка при виклику зі строковим аргументом `str`, буде повертати рядок з усіма символам в `str`, зміненими в верхній регістр.

Функції викликаються з вказівкою аргументів в дужках. Приклади:

- `$uppercase("Hello")` повертає рядок "HELLO".
- `$substring("hello world", 0, 5)` повертає рядок "hello"
- `$sum([1,2,3])` повертає число 6

Анонімна (лямбда) функція може бути означена, використовуючи наступний синтаксис:

```
function($l, $w, $h){ $l * $w * $h }
```

і може бути викликана з допомогою:

```
function($l, $w, $h){ $l * $w * $h }(10, 10, 5)
```

що поверне результат 500

Функція може бути назначена змінній для наступного використання (всередині блоку):

```
(
  $volume := function($l, $w, $h){ $l * $w * $h };
  $volume(10, 10, 5);
)
```

Функції можуть бути означені за допомогою додаткових підписів, яка означують типи параметрів функції. Якщо це передбачено, перш ніж функція буде викликана, будуть перевірятися аргументи. Якщо список аргументів не відповідає підпису, виникне динамічна помилка виникає.

Сигнатура функції це рядок в формі `<params:return>` . `params` – це послідовність символів типу, кожен з яких представляє тип вхідних аргументів. `return` – це символ типу, що представляє тип вихідного значення функції. Нижче наведені типи:

Прості типи:

- `b` - Boolean
- `n` - number
- `s` - string
- `l` - `null`

Складені типи:

- `a` - array
- `o` - object
- `f` - function

Типи об'єднання:

- `(sao)` - string, array або object
- `(o)` – те саме що і `o`
- `u` - еквівалентно `(bnsl)` тобто Boolean, number, string або `null`
- `j` – будь-який тип JSON. Еквівалентно `(bnsloa)` тобто Boolean, number, string, `null`, object або array, але не function
- `x` - будь-який тип, Еквівалентно `(bnsloaf)`

Параметричні типи:

- `a<s>` - масив рядків
- `a<x>` - масив значень будь-яких типів

Декілька прикладів сигнатур вбудованих функцій JSONata:

- `$count` має сигнатуру `<a:n>`; приймає масив і повертає число.
- `$append` має сигнатуру `<aa:a>`; приймає два масиви і повертає масив.
- `$sum` має сигнатуру `<a<n>:n>`; приймає масив чисел і повертає число.
- `$reduce` має сигнатуру `<fa<j>:j>`; приймає редукуючу функцію `f` і `a<j>` (масив об'єктів JSON objects) і повертає об'єкт JSON.

Кожен тип символу може також мати *options*.

- `+` - один або більше аргументів цього типу
  - E.g. `$zip` has signature `<a+>`; it accepts one array, or two arrays, or three arrays, or...
- `?` – опійний аргумент
  - E.g. `$join` has signature `<a<s>s?:s>`; it accepts an array of strings and an optional joiner string which defaults to the empty string. It returns a string.
- `-` - якщо аргумент відстуній, використовується значення контексту ("focus").
  - E.g. `$length` has signature `<s-:n>`; it can be called as `$length(OrderID)` (one argument) but equivalently as `OrderID.$length()`.

Функції, призначені для змінних, можуть викликати себе, використовуючи це посилання змінної. Це дозволяє визначити рекурсивні функції. Наприклад.

JSONata

```
(
  $factorial:= function($x){ $x <= 1 ? 1 : $x * $factorial($x-1) };
  $factorial(4)
)
```

Result `24`

Зауважимо, що фактично можна написати рекурсивну функцію, використовуючи суто анонімні функції (тобто нічого не присвоюється змінним). Це робиться за допомогою комбінатора Y, який може бути цікавим викликом для тих, хто цікавиться функціональним програмуванням.

Більше про функції можна прочитати за цим посиланням.

Використання регулярних виразів

Про регулярні вирази можна прочитати за цим посиланням.

Робота з датою та часом

Є дві функції, які повертають відмітку часу:

1. `$now()` повертає відмітку часу в форматі рядку ISO 8601.
2. `$millis()` повертає ту саму відмітку часу як кількість мілісекунд починаючи з опівночі 1-го січня 1970 UTC (the Unix epoch).

Відмітка часу фіксується з початку оцінювання виразу, і та сама відмітка часу повертається для кожного входження `$now()` або `$millis()` в тому самому виразі протягом тривалості оцінювання. Наприклад:

JSONata
```
{
  "invoiceTime": $now(),
  "total": $sum(Account.Order.Product.(Price * Quantity)),
  "closingTime": $now()
}
```

Result
```
{
  "invoiceTime": "2018-12-10T13:49:51.141Z",
  "total": 336.36,
  "closingTime": "2018-12-10T13:49:51.141Z"
}
```

```
ISO 8601 формат:
{
    "myDateTime": "2018-12-10T13:45:00.000Z"
}
```

JSONata дотримується цієї конвенції та надає функції для форматування та розбору відміток часу ISO 8601 (`toMillis()` and `fromMillis()`). Приклади

JSONata `$toMillis('10/12/2018', '[D]/[M]/[Y]') ~> $fromMillis('[M]/[D]/[Y]')`

Result `"12/10/2018"`

JSONata
```
$toMillis('10/12/2018', '[D]/[M]/[Y]')
        ~> $fromMillis('[FNn], [D1o] [MNn] [YI]')
```

Result `"Monday, 10th December MMXVIII"`

Оператори

Navigation Operators

- `.` (Map)
- `[...]` (Filter)
- `~>` (Chain)
- `^(...)` (Order-by)
- `... ~> | ... | ... |` (Transform)
- `&` (Concatenation)
- `? :` (Conditional)
- `:=` (Variable binding)

Numeric Operators

- `+` (Addition)
- `-` (Substraction/Negation)
- `*` (Multiplication)
- `/` (Division)
- `%` (Modulo)
- `..` (Range)

Comparison Operators

- `=` (Equals)
- `!=` (Not equals)
- `>` (Greater than)
- `<` (Less than)
- `>=` (Greater than or equals)
- `<=` (Less than or equals)
- `in` (Inclusion)

Boolean Operators

- `and` (Boolean AND)
- `or` (Boolean OR)

Бібліотека функцій

String functions

- `$string()`
- `$length()`
- `$substring()`
- `$substringBefore()`
- `$substringAfter()`
- `$uppercase()`
- `$lowercase()`
- `$trim()`
- `$pad()`
- `$contains()`
- `$split()`
- `$join()`

- $match()
- $replace()
- $eval()
- $base64encode()
- $base64decode()

Numeric functions

- $number()
- $abs()
- $floor()
- $ceil()
- $round()
- $power()
- $sqrt()
- $random()
- $formatNumber()
- $formatBase()
- $formatInteger()
- $parseInteger()

Numeric aggregation functions

- $sum()
- $max()
- $min()
- $average()

Boolean functions

- $boolean()
- $not()
- $exists()

Array Functions

- $count()
- $append()
- $sort()
- $reverse()
- $shuffle()
- $zip()

Object functions

- $keys()
- $lookup()
- $spread()
- $merge()
- $sift()
- $each()

Date/Time functions

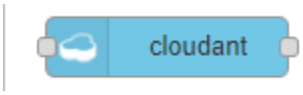
1. `$now()`
2. `$millis()`
3. `$fromMillis()`
4. `$toMillis()`

Higher order functions

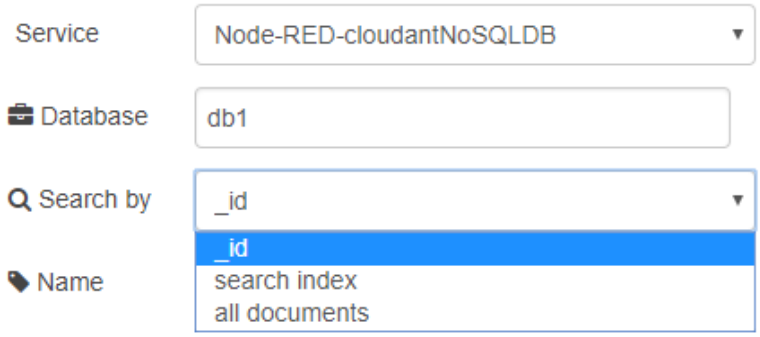
- `$map()`
- `$filter()`
- `$reduce()`
- `$sift()`

Storage Cloudant

cloudant in



Вузол що дозволяє отримувати документи з бази даних Cloudant.



Пошук документів може проводитися в трьох режимах: безпосередньо по **_id**, використовуючи існуючий Пошуковий індекс або отримуючи усі документи, збережені в базі даних.
1) Якщо використовується запит з опцією **_id**, значення для документа `_id` повинно бути вставлене `msg.payload` як string.
2) Для використання існуючого **Пошукового індексу,** що збережений в потрібній базі даних, аргумент пошуку повинен бути переданий в `msg.payload` як string за шаблоном `indexName:value`. Майте на увазі, що індекс повинен бути створений заздалегідь в базі даних, тут посилаються на його `design document/index name`.

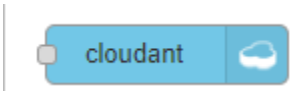
При запиті за допомогою пошукового індексу ви можете передавати параметри пошуку як об'єкт у `msg.payload`. Наприклад, ви можете передати такий об'єкт:

```
{ query: "abc*", limit: 100 }
```

для зміни значення `limit`. Ви можете знайти більше інформації про пошуковий індекс офіційній документації Cloudant.

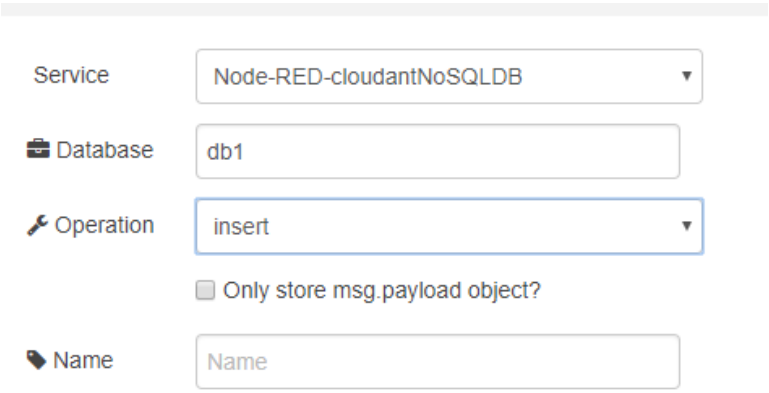
3) Останній спосіб отримання документів полягає в тому, щоб просто отримати їх всі, вибравши опцію **all documents.**
Ім'я бази даних повинно відповідати цим правилам:
• Без пробілів
• Всі літери маленькі
• Перший символ не може бути _

cloudant out



Простий вузол виводу в Cloudant зберігає вхідне повідомлення **msg** у вказаній базі даних. Якщо виставлена опція «Only story msg.payload object», то зберігатиметься не все повідомлення, а тільки корисне навантаження.

Якщо параметр `_id` не вказаний, Cloudant генерує випадковий id і кожного разу буде вставлений новий документ.



Якщо Ви хочете **оновити** існуючий документ, ви повинні вказати останні значення для `_id` та `_rev` у вашому об'єкті. Якщо вони не відповідають поточним значенням, що зберігаються в базі даних, ви отримаєте повідомлення про помилку конфлікту, тому переконайтеся, що ви використовуєте найновішу інформацію для документа.

Також можна видаляти документи з бази даних, надаючи значення `_id` та `_rev` і вибираючи опцію "**remove**" для вузла. Ви можете передавати ці значення в самому об'єкті msg або як об'єкт в `msg.payload`.

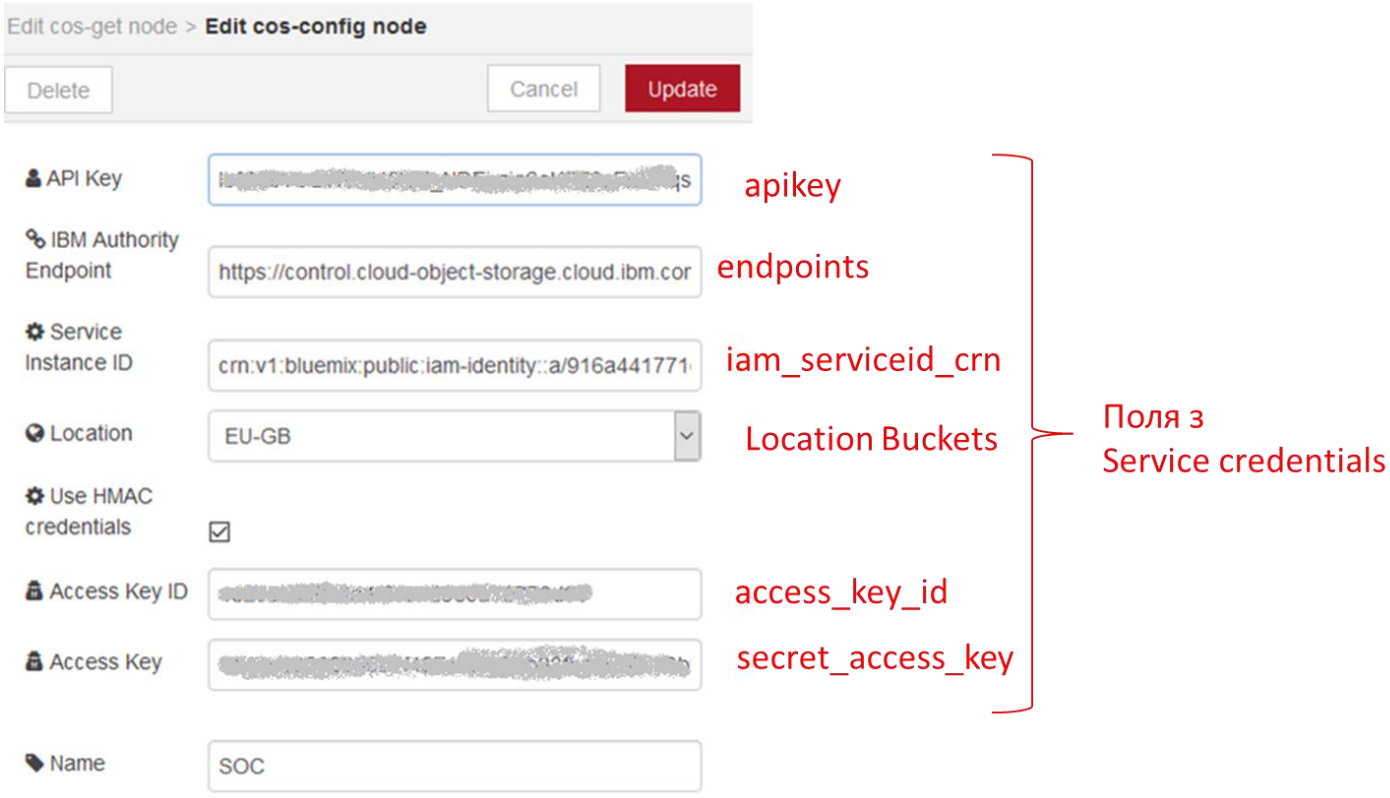
Ім'я бази даних повинно відповідати наступним правилам:

- Без пробілів
- Всі літери маленькі
- Перший символ не може бути `_`

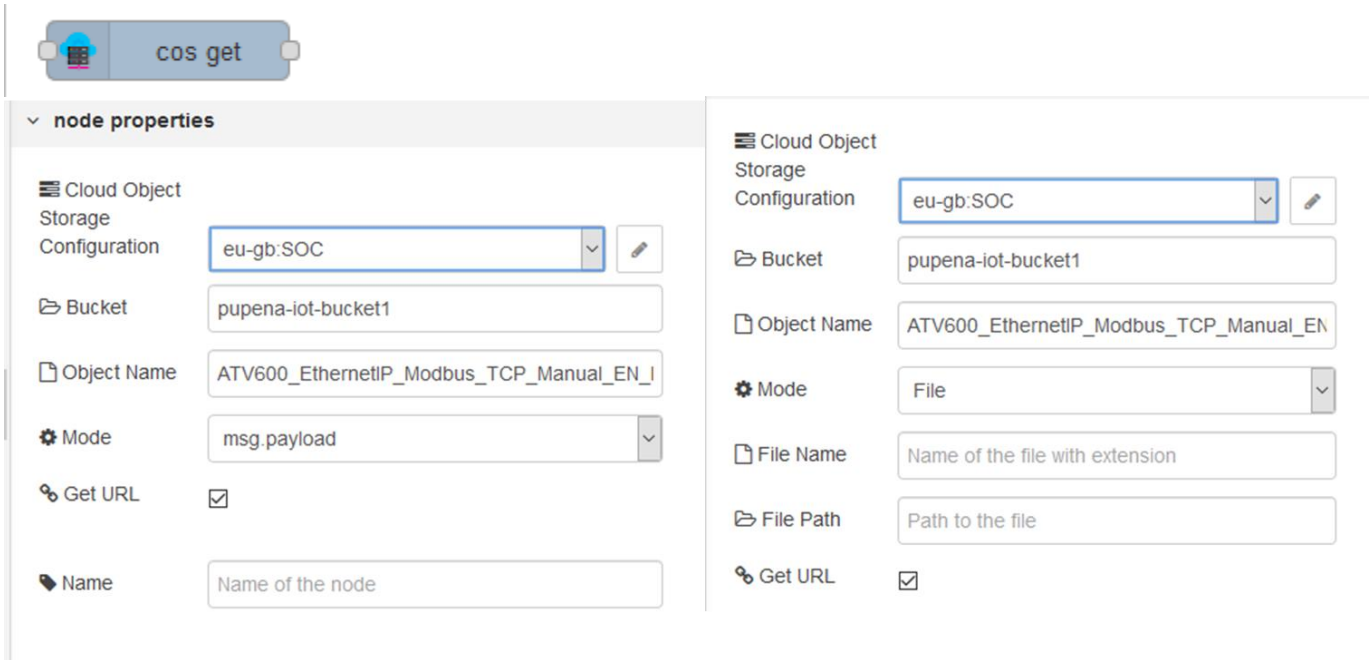
У вашому документі слід уникати полів верхнього рівня, які починаються з `_`, за винятками для `_id`, `_rev` та інших зарезервованих слів CouchDB reserved words.

Storage COS (node-red-contrib-cos)

cos config



cos get



Вузол для отримання (GET) об’єкту з IBM Cloud Object Storage Service (S3).
Входи:
payload (string | buffer) – корисне навантаження, що ініціює читання
bucket (string) - bucket в якому знаходиться потрібний об’єкт

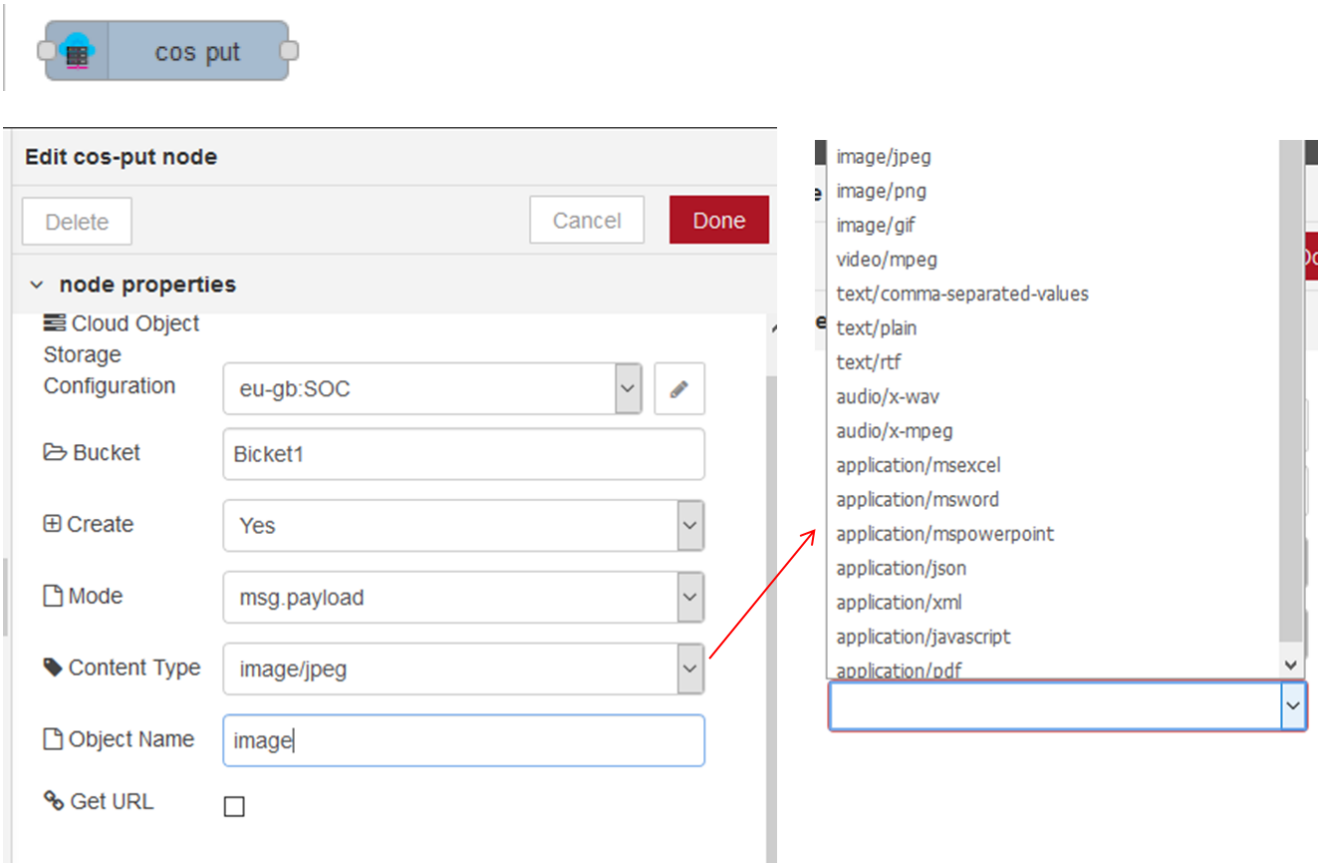
objectname (string) – унікальне ім’я збереженого файлу в Cloud Object Storage Service.
mode (string) – вказує чи зберігати об’єкт в файл.
filename (string) – ім’я файлу для перезапису.
filepath (string) – ім’я папки з файлом .
geturl (Boolean) – якщо необхідно отримати URL об’єкту, тоді вказується `true`.

Виходи.

1. Стандартні
   payload (string) – вузол повертає об’єкт в `msg.payload` або в `filename` якщо вибраний file-mode.
   objectname (string) – Ім’я об’єкту `msg.objectname`.
   url (string) – Якщо виставлена опція `msg.geturl`, то `msg.url` буде містити WebLink URL на об’єкт.
2. Помилка
   error (string) – або `msg.error` з кодом помилки.

Поле `msg.objectname` повинно містити унікальне ім’я об’єкту який необхідно отримати з IBM Cloud Object Storage Service
Якщо ви хочете отримати згенерований URL для об’єкту, необхідно виставити опцію HMAC.
Цей вузол використовує функцію S3-API IBM Cloud Object Storage Service.

cos put



Цей вузол відправляє об’єкт на збереження в IBM Cloud.

Входи

payload (string | buffer) – корисне навантаження повідомлення (вміст об’єкту), що ініціює відправку.
bucket (string) – існуючий bucket, в який буде збережено об’єкт.

create (string) – назва bucket, що потрібно створити, якщо він не існує. Опція Create вказує чи потрібно створювати новий bucket.
objectname (string) – унікальне ім'я збереженого файлу в Cloud Object Storage Service.
mode (string) – показує звідки завантажувати вміст об'єкту, з файлу чи з корисного навантаження.
filename (string) – ім'я файлу, якщо завантаження відбувається з файлу
filepath (string) – шлях до файлу, , якщо завантаження відбувається з файлу.
geturl (Boolean) якщо необхідно отримати URL об'єкту, тоді вказується `true`.

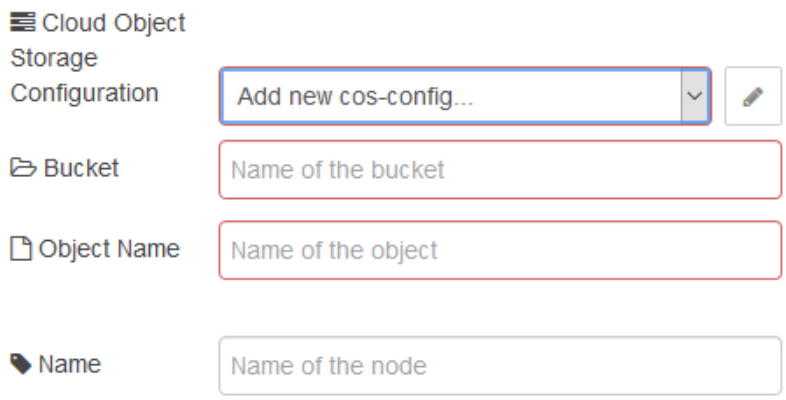
Виходи:
Стандартний вихід:
payload (string) – вузол повертає об'єкт `msg.payload`, або записує в `filename`, якщо використовується режим збереження в файл.
objectname (string) – ім'я об'єкту `msg.objectname`.
url (string) – Якщо `msg.geturl` встановлено, `msg.url` буде містити WebLink URL на об'єкт.

Вихід при помилці:
error (string) – код помилки.

Поле `msg.payload` вміщує завантажений файл, або у випадку режиму запису в файл є тільки тригером, що показує на те, що файл записано.
Параметри налаштування вузла `msg.filename`, `msg.filepath` та `msg.fileformat` є типово встановленими значеннями і можуть бути переписані відповідними полями вхідного повідомленнями.
Вузол повертає `msg.payload` і назву об'єкту `msg.objectname` або `msg.error` з кодом помилки.
Якщо необхідно отримати URL об'єкту, що генерується, необхідно активувати поле HMAC.
Вузол використовує S3-API функції IBM Cloud Object Storage Service.

cos del

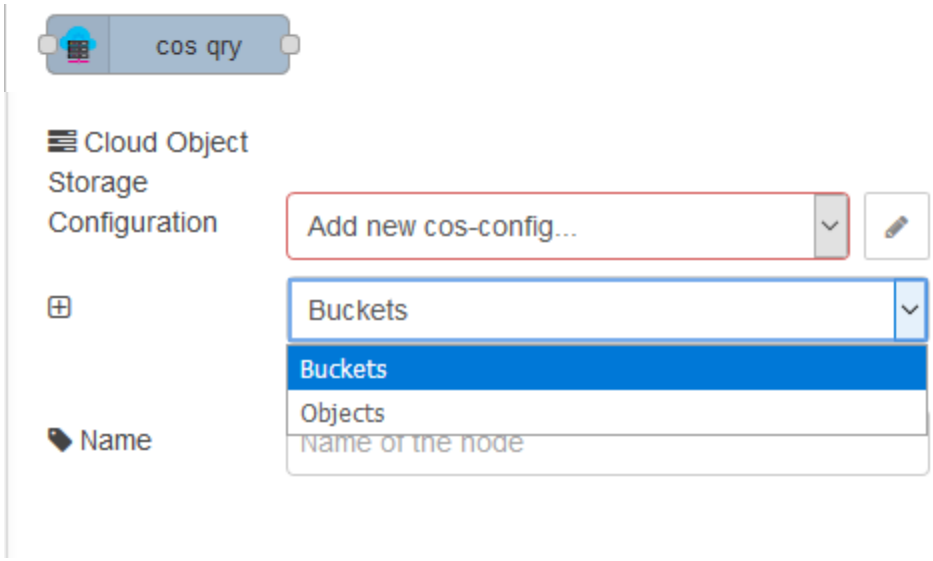
Цей вузол видаляє об'єкт з IBM Cloud.



Входи:
payload (string | buffer) – корисне навантаження повідомлення що ініціює видалення
bucket (string) – існуючий bucket, в якому знаходиться об'єкт, що необхідно видалити.
objectname (string) – унікальне ім'я, що необхідно видалити.

Вузол повертає `msg.status` , якщо об’єкт видалено, або `msg.error` з помилкою, якщо не вдалося видалити.

cos qry



Цей вузол показує формує запит на отримання списку buckets або об’єктів в IBM Cloud.

Входи:
payload (string | buffer) – корисне навантаження повідомлення, що ініціює запит.
list (string) – вибір на отримання списку всіх buckets або всіх об’єктів в bucket.
bucket (string) – існуючий bucket, з якого треба отримати список об’єктів.

Вузол повертає `msg.payload` і ім’я об’єкту `msg.objectname` або `msg.error` з помилкою errorcode.

Watson IoT Device/Gateway (node-red-contrib-ibm-watson-iot)

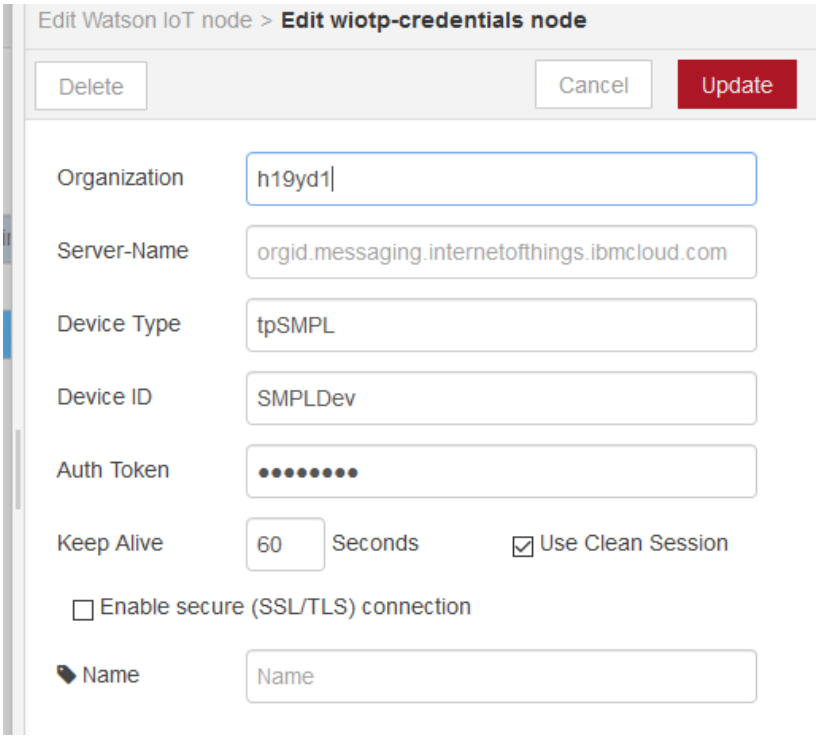
Вузли даної бібліотеки Node-RED призначені для підключення до платформи IBM Watson Internet of Things в якості пристрою або шлюзу з використанням протоколу MQTT. Слід відмітити, що для підключення до IBM Watson IoT можна використовувати і бібліотеку вузлів MQTT, однак це не так зручно. Окрім MQTT для підключення Node-RED до Watson IoT Platform в якості пристроїв чи шлюзів можна використовувати HTTP API.

Додатково про підключення до Watson IoT Platform з боку пристроїв та шлюзів можна прочитати за наступним посиланнями:

- MQTT messaging
- MQTT connectivity for devices
- MQTT connectivity for gateways
- HTTP APIs for devices
- HTTP APIs for gateway devices
- Device Management Protocol
- Device management requests
- Extending device management

wiotp credentionals (конфігураційний вузол означення повноважень доступу)

Конфігурування повноважень для доступу до двійників пристроїв в IBM Watson IoT, відбувається в даному конфігураційному вузлі.



У полі Organization вказується ORG ID IBM Watson IoT. Його можна побачити в правому верхньому кутку консолі налаштування сервісу IBM Watson IoT або у відповідному полі вкладки Settings.

У полі Server-Name вказується ім'я серверу. Імя серверу вказується в форматі:

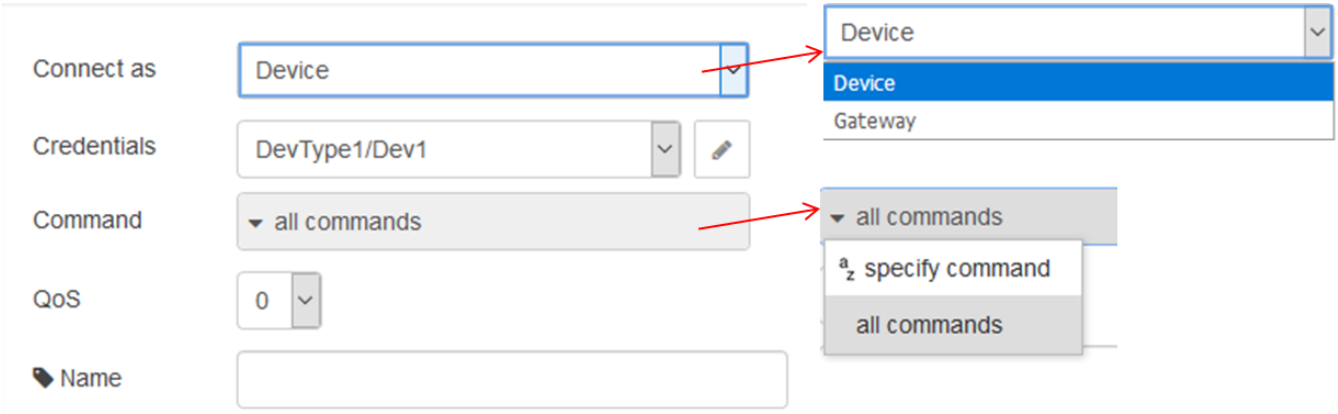
org_id.messaging.internetofthings.ibmcloud.com

де *org_id* – це Ваш ORG ID IBM Watson IoT. Якщо це поле залишити порожнім Server-Name сформується автоматично. Слід зазначити що так само формується ім'я серверу у випадку використання вузлів MQTT. Додатково про налаштування зв'язку по MQTT можна прочитати за цим посиланням.

У Device Type та Device ID вказуються назви типів та пристроїв, які означені в Watson IoT Platform. При створенні пристрою йому надається маркер доступу, Auth Token, який вказується у однойменному полі. Слід зазначити, що при створенні пристрою, Auth Token необхідно кудись записати, бо потім він буде недоступним для перегляду. Інші налаштування аналогічні як у вузла Mqtt-broker.

wiotp in

Отримує команди для пристроїв/шлюзів з платформи IBM Watson Internet of Things.



Вузол може підключатися як пристрій (Device) або шлюз(Gateway):

- Пристрій(**Device**): вузол може бути налаштований на отримання всіх команд для пристрою або тільки вибраного типу.
- Шлюз(**Gateway**): вузол може бути налаштований на прийом команд для всіх пристроїв, підключених через шлюз, або для вибору підмножини з них.

Повідомлення, надіслане цим вузлом, буде містити такі властивості:

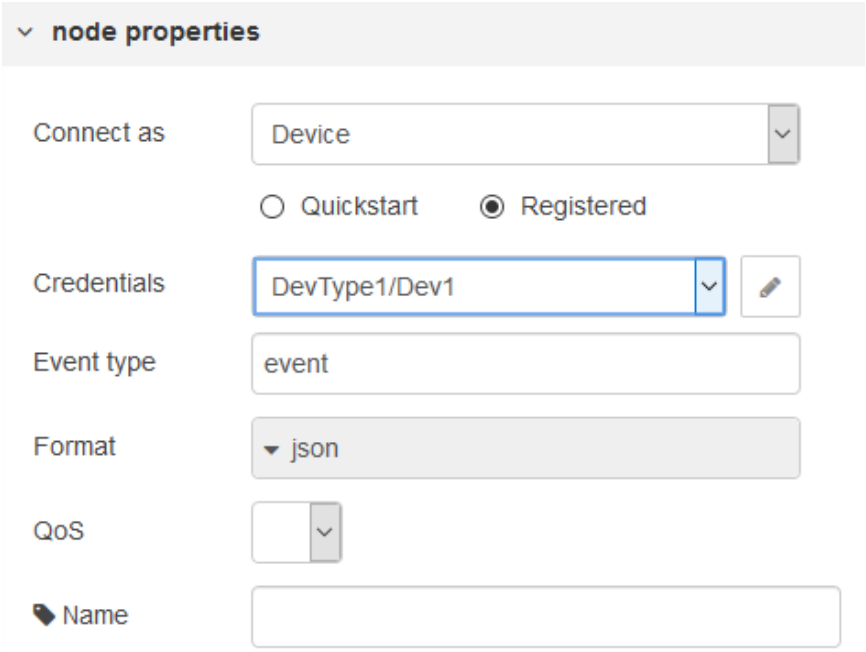
- `payload` - тіло команди. Якщо команда була ідентифікована як json, ця властивість буде об'єктом JavaScript, інакше це буде string.
- `topic` - тема, на яку було отримано команду
- `command` - ім'я команди
- `format` - формат команди
- `deviceType` - (тільки для шлюзу) типу пристрою, для якого призначена команда
- `deviceId` - (тільки для шлюзу) ідентифікатора пристрою, для якого призначена команда

- `server-name` - (необов'язково) ім'я кінцевої точки обміну повідомленнями Watson IoT Platform. За замовчуванням це значення буде рівним: "[organization id].messaging.internetofthings.ibmcloud.com" (Watson IoT Production server). Властивість mqtt_host облікових даних служби Watson IoT несе значення імені кінцевої точки обміну повідомленнями.

Можуть бути вказані значення для QoS (Quality of Service) і значення keep alive interval (див.MQTT).

wiotp out

Надсилає події пристрою до платформи IBM Watson Internet of Things.



Вузол може підключатися як пристрій (Device) так і як шлюз (Gateway). він може працювати в режимі «registered» або як «Quickstart». При підключенні в режимі Quickstart з'єднання буде проводитися з існуючим тестовим сервером, використовувати тип пристрою `node-red-wiotp` і вказаний ідентифікатор пристрою. Події з вузла можуть бути переглянуті на панелі інструментів тестового серверу Quickstart dashboard.

Тип відправленої події (event type) може бути налаштований у вузлі або, якщо залишений порожнім, може бути встановлений властивістю `msg.event`. У випадку, якщо використовується сервіс Quickstart, тип події повинен бути встановлений у значення `event`.

Формат події типово є `json`, але може бути встановлений на інше значення або, якщо він залишений порожнім, може бути встановлений властивістю `msg.format`.

Дані для події беруться з `msg.payload`. Якщо формат встановлено на `json`, цей вузол намагатиметься відповідним чином кодувати дані:

- Якщо дані є об'єктом форми: `{ d: { ... }}` вони будуть передаватися у тому ж самому вигляді. Аналогічно, якщо це строкове представлення такого об'єкта, подальшого кодування не буде зроблено.
- Для будь-якого іншого типу об'єкта, наприклад числового, значення буде перетоврену в вигляд `{"d":{"value":123}}`

Така структура повідомлення пристрою продиктована форматом означеним в IBM Cloud, короткий опис доступний за цим посиланням.

Якщо `format` встановлено в значення other, дані будуть передаватися в такому вигляді як приходять на вхід.

При підключенні в якості шлюзу, тип і ідентифікатор Device, на який надсилається подія від його імені, може бути налаштований у вузлі або, якщо поле залишено порожнім, може бути встановлений властивостями `msg.deviceType` і `msg.deviceId`. Якщо ці властивості не надаються, або в вузлі, або в повідомленні, вузол буде використовувати тип і ідентифікатор самого шлюзу.

Інші налаштування аналогічні як в вузлах MQTT.

IBM IoT APP (node-red-contrib-scx-ibmiotapp)

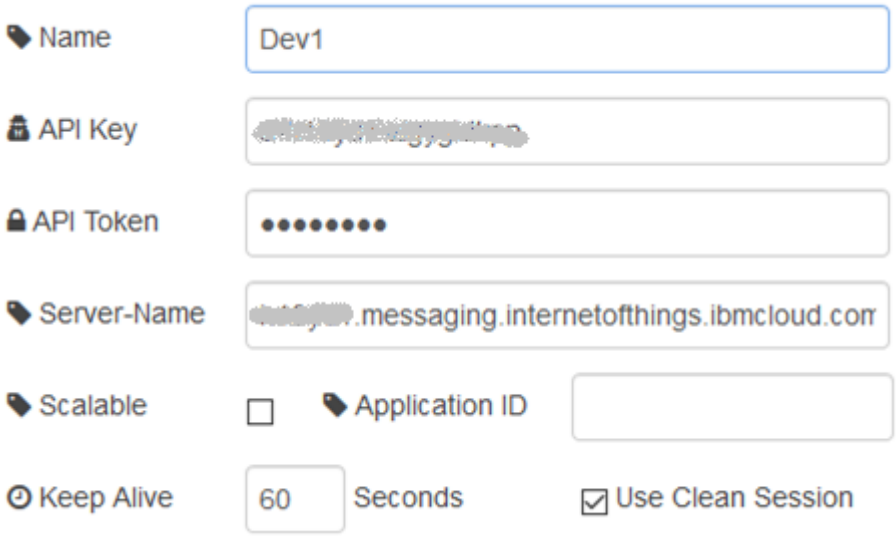
Вузли даної бібліотеки Node-RED призначені для підключення до платформи IBM Watson Internet of Things в якості застосунку (Application) з використанням протоколу MQTT. Слід відмітити, що для підключення до IBM Watson IoT можна використовувати і бібліотеку вузлів MQTT. Окрім MQTT для підключення Node-RED до Watson IoT Platform в якості застосунку можна використовувати HTTP API.

Додатково про підключення до Watson IoT Platform з боку застосунків можна прочитати за наступним посиланнями:

- HTTP REST API for applications
- MQTT messaging
- MQTT connectivity for applications

ibmiot config

У даному вузлі вказується точка доступу до API, означеного в IBM Watson IoT Platform в розділі APPs.



У полі API Key вказується відповідне значення Key (див. Browse API Keys в IoT Platform), а в полі API Token – маркер, що був отриманий при створенні. Слід зазначити, що маркер при створенні точки доступу API треба зберегти, бо він не буде видимий пізніше.

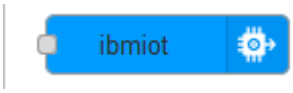
У полі Server-Name необхідно вказати:

org_id.messaging.internetofthings.ibmcloud.com

де *org_id* – це Ваш ORG ID IBM Watson IoT. Його можна побачити в правому верхньому кутку консолі налаштування сервісу IBM Watson IoT або у відповідному полі вкладки Settings.

Application ID – необов’язковий ідентифікатор застосунку, від імені якого відбувається з’єднання. Якщо він не вказаний, то генерується автоматично.

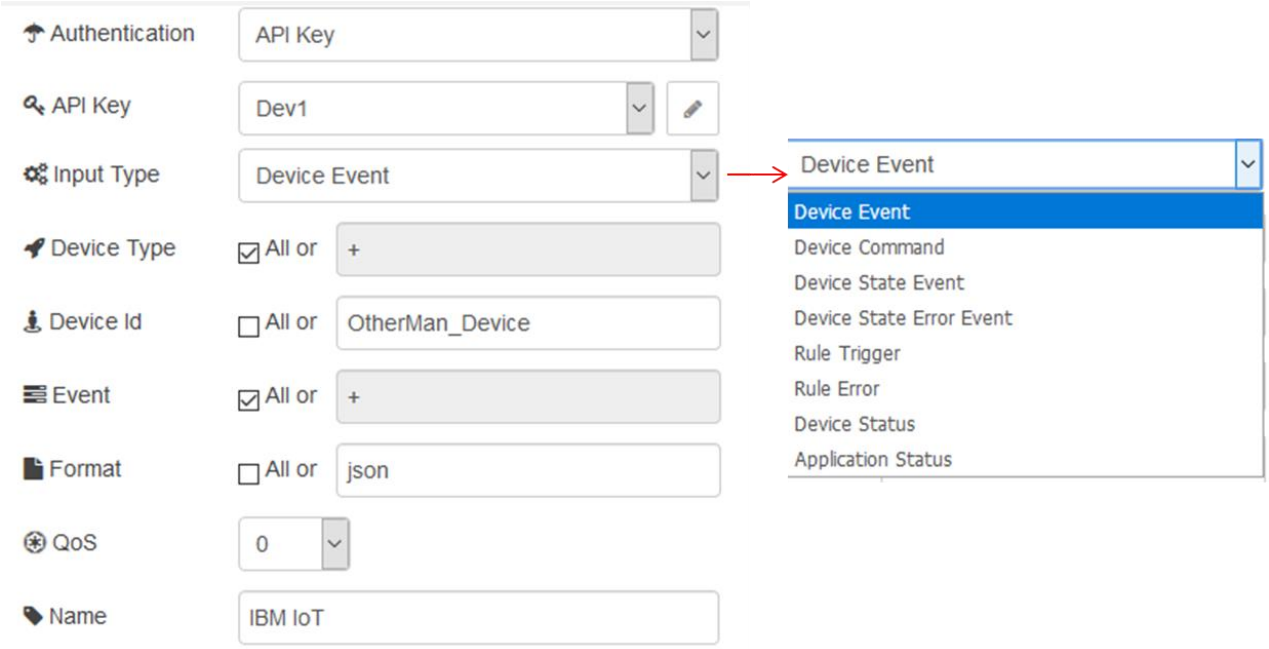
ibmiot in



Вхідний вузол, який можна використовувати з Watson IoT Platform для отримання подій, надісланих пристроями, прийому команд, надісланих на пристрої, або отримання оновлень статусу щодо пристроїв або застосунків.

Вузол реалізує API MQTT, який детально описаний за наступним посиланням [MQTT connectivity for applications]

Він створює об'єкт msg, і встановлює **msg.payload** як рядок, що містить корисне навантаження вхідного повідомлення.

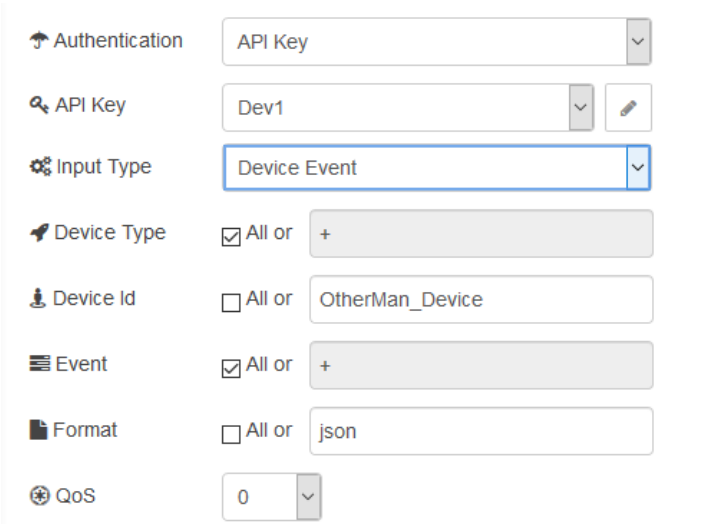


Поле Authentication вказується тип серверу для з'єднання. Вузол може підключатися як по вказаному «API Key» так і до наперед визначеного текстового «Quickstart». При підключенні в режимі Quickstart з'єднання буде проводитися для тестування з існуючим тестовим сервером, використовувати тип пристрою `node-red-wiotp` і вказаний ідентифікатор пристрою. Події з вузла можуть бути переглянуті на панелі інструментів тестового серверу Quickstart dashboard. У випадку API Key вказується наперед сконфігурований API Key.

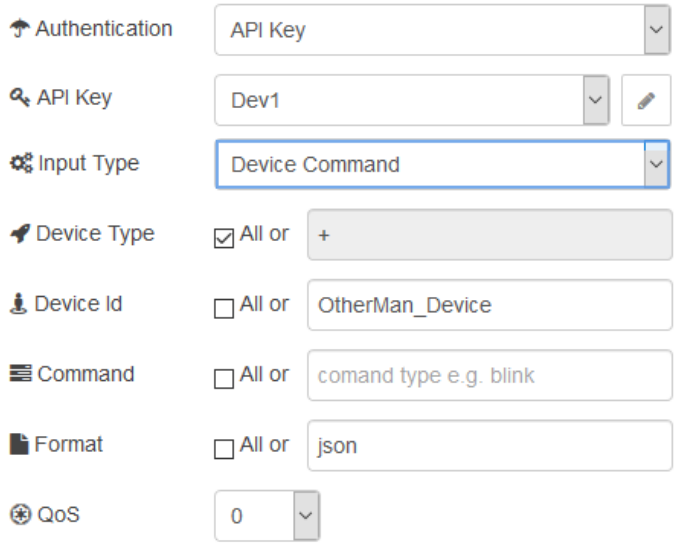
Поле Input Type може приймати кілька значень.

1) Значення **Device Event** вказує на необхідність отримання подій, які означені типом пристрою Device Type, його ідентифікатором Device ID, та типом події Event. У випадку, якщо необхідно виділити поля будь-якого типу, необхідно виставити опцію «All» (відповідник «+»).

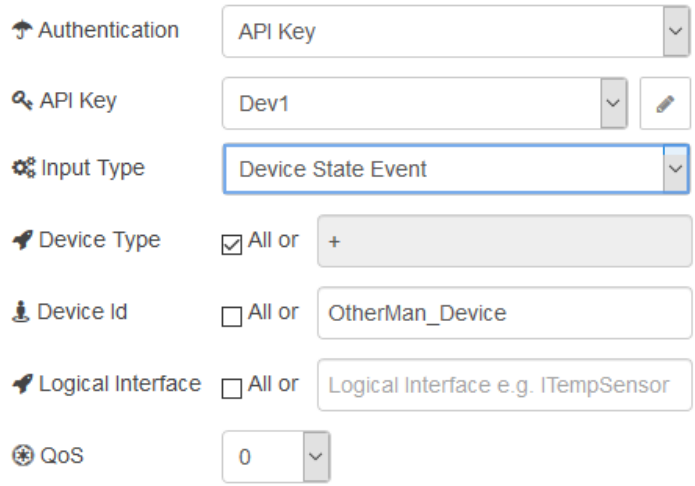
Значення "Формат" зберігається в **msg.format**. Цей вузол підтримує формати json, buffer та інші типи. Коли формат встановлено в json, цей вузол аналізує вхідні дані за допомогою JSON.parse () і і формує результат на вихід. Якщо цим вузлом отриманий об'єкт buffer, він виводить вміст без будь-якого перетворення. А для інших типів цей вузол виводить повідомлення в форматі String.



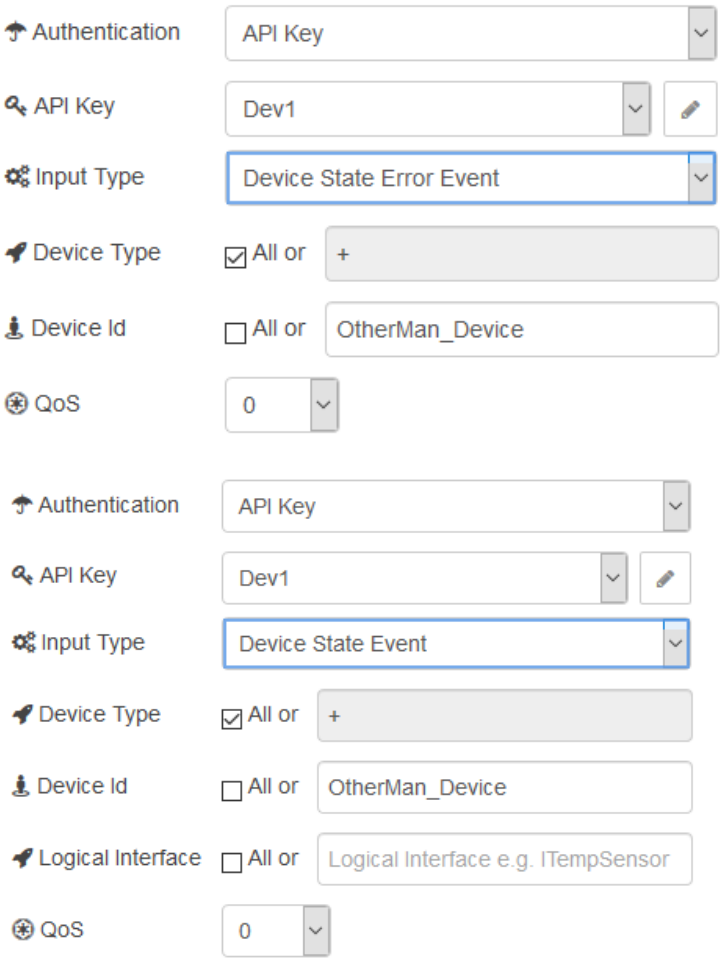
2) Значення типу Device Command вказує на необхідність отримання усіх команд, визначених полем «Command» для вказаних пристроїв.



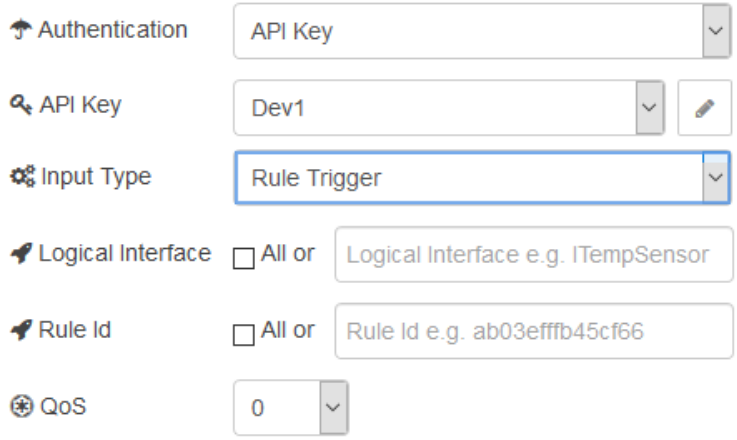
3) Значення типу Device State Event вказує на необхідність отримання стану усіх пристроїв через їх означені логічні інтерфейси, визначених полем «Logical Interface» для вказаних пристроїв.



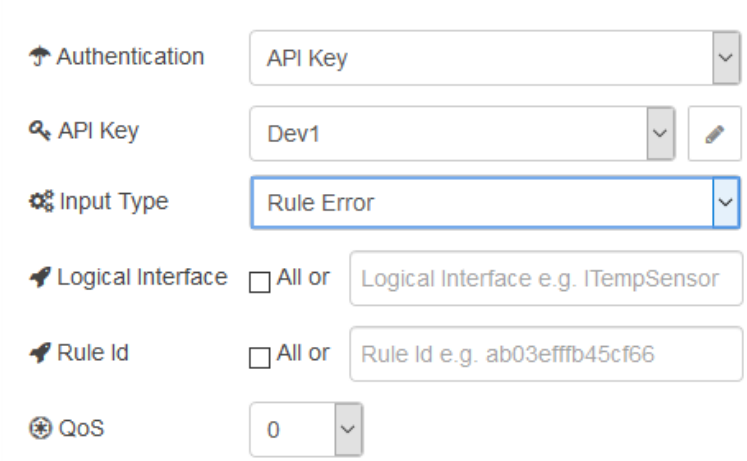
4) Значення типу Device State Error Event вказує на необхідність отримання стану усіх помилок для вказаних пристроїв.



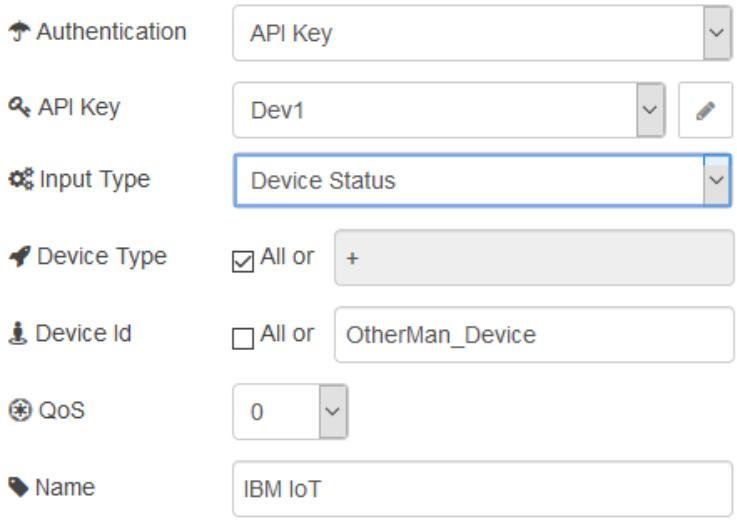
5) Значення типу Rule Trigger вказує на необхідність отримання подій означених вказаними правилами (Rule ID) для логічного інтерфейсу пристрою.



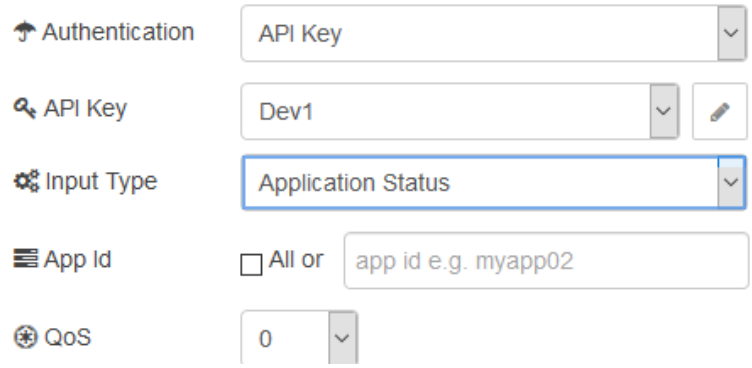
6) Значення типу Rule Error вказує на необхідність отримання помилок означених вказаними правилами (Rule ID) для логічного інтерфейсу пристрою.



7) Значення типу Device Status вказує на необхідність отримання статусу вказаного пристрою.



8) Значення типу Application Status вказує на необхідність отримання статусу вказаного застосунку.

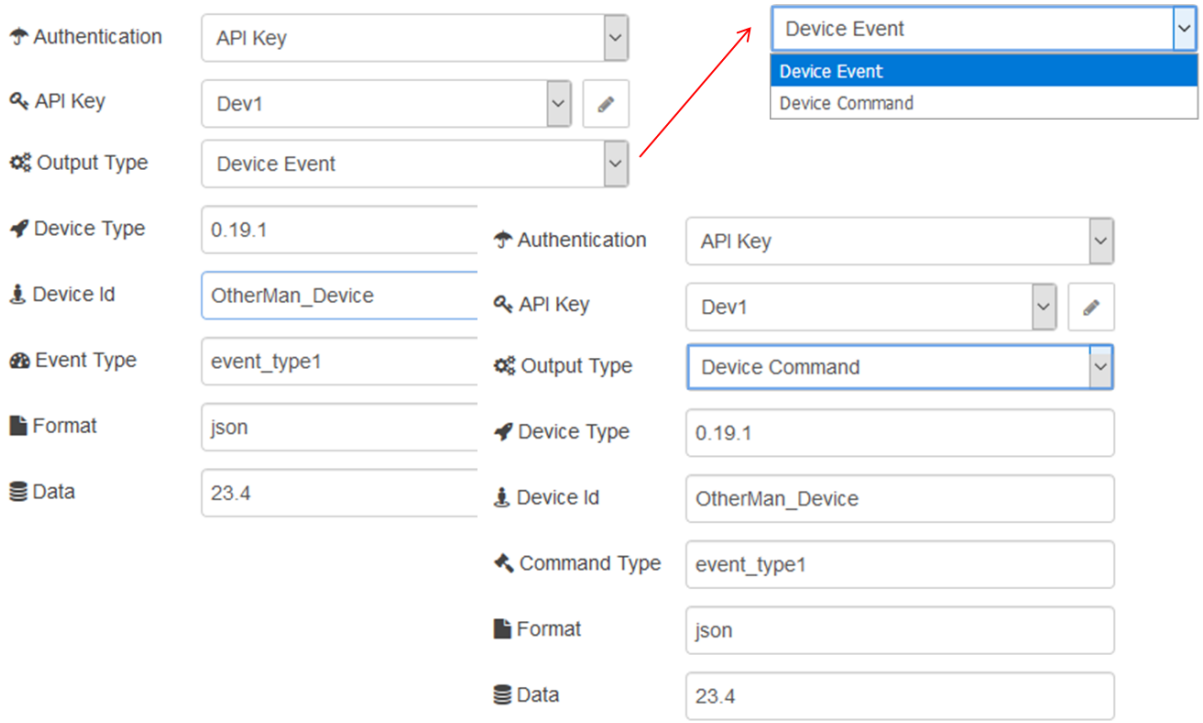


Ряд значень можна задавати через поля повідомлень:

- "Device Id" зберігається в **msg.deviceId**
- "Application Id" зберігається в **msg.applicationId**
- "Device Type" зберігається в **msg.deviceType**
- "Event Type" зберігається в **msg.eventType**
- "Command Type" зберігається в **msg.commandType**

ibmiot out

Вихідний вузол, який можна використовувати з Watson IoT Platform для надсилання команд на пристрій або надсилання події від імені пристрою. Вузол реалізує API MQTT, який детально описаний за наступним посиланням MQTT connectivity for applications
Тип команди (Output type = Device Command) чи події (Output type = Device Event) вказується відповідно в полях Command Type та Device Type.



Наступні властивості повідомлення мають пріоритет і замінюють значення, налаштовані в налаштуваннях вузлу:

- **msg.deviceId** замінює значення " Device Id "
- **msg.deviceType** замінює значення " Device Type "
- **msg.eventOrCommandType** замінює значення " Event Type " або "Command Type"
- **msg.format** замінює значення "Format".

Цей вузол підтримує json, buffer та інші типи. Якщо формат встановлений в json, цей вузол очікує тип object, або версію Stringify об'єкта json. Для того, щоб відправити buffer або інші типи, просто надішліть вміст без будь-яких перетворень і встановіть відповідний формат.

- **msg.payload** перевизначає значення "Дані"

Посилання для підготовки та обговорення

1. Форум АСУ в Україні
2. Група у ФБ АСУ в Україні
3. Матеріали дистанційного курсу Технології Індустрії 4.0.
4. Лабораторна робота №1. Основи роботи з Node-RED.
5. Лабораторна робота №2. Протоколи IoT. Частина 1. MQTT.
6. Лабораторна робота №2. Протоколи IoT. Частина 2. Використання WEB API та Web-сокетів.
7. Лабораторна робота №3. Хмарні сервіси для Індустрії 4.0. Частина 1. Основи роботи з хмарною платформою IBM Cloud.
8. Лабораторна робота №3. Хмарні сервіси для Індустрії 4.0. Частина 2. Хмарні сервіси для збереження об’єктів.
9. Лабораторна робота №3. Хмарні сервіси для Індустрії 4.0. Частина 3. Основи роботи з Cloud Foundry в IBM Cloud.
10.