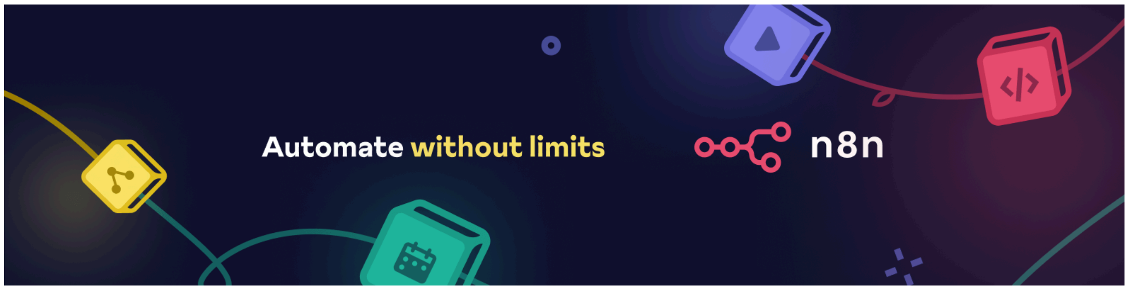
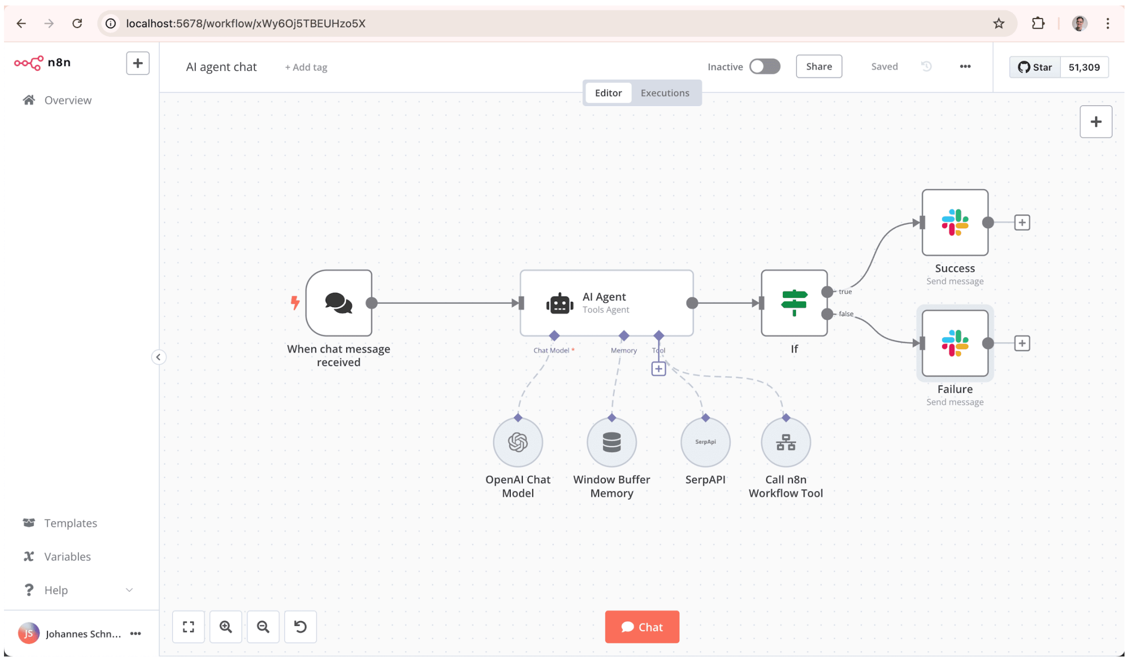
README.md



Banner image

n8n - Secure Workflow Automation for Technical Teams

n8n is a workflow automation platform that gives technical teams the flexibility of code with the speed of no-code. With 400+ integrations, native AI capabilities, and a fair-code license, n8n lets you build powerful automations while maintaining full control over your data and deployments.



n8n.io - Screenshot

Key Capabilities

- **Code When You Need It**: Write JavaScript/Python, add npm packages, or use the visual interface

- **AI-Native Platform**: Build AI agent workflows based on LangChain with your own data and models
- **Full Control**: Self-host with our fair-code license or use our cloud offering
- **Enterprise-Ready**: Advanced permissions, SSO, and air-gapped deployments
- **Active Community**: 400+ integrations and 900+ ready-to-use templates

Quick Start

Try n8n instantly with npx (requires Node.js):

```
npx n8n
```

Or deploy with Docker:

```
docker volume create n8n_data
docker run -it --rm --name n8n -p 5678:5678 -v n8n_data:/home/node/.n8n
docker.n8n.io/n8nio/n8n
```

Access the editor at http://localhost:5678

Resources

- 📚 Documentation
- 🔧 400+ Integrations
- 💡 Example Workflows
- 🤖 AI & LangChain Guide
- 👥 Community Forum
- 📖 Community Tutorials

Support

Need help? Our community forum is the place to get support and connect with other users: community.n8n.io

License

n8n is fair-code distributed under the Sustainable Use License and n8n Enterprise License.

- **Source Available**: Always visible source code
- **Self-Hostable**: Deploy anywhere
- **Extensible**: Add your own nodes and functionality

Enterprise licenses available for additional features and support.

Additional information about the license model can be found in the docs.

Contributing

Found a bug 🐛 or have a feature idea ✨? Check our Contributing Guide to get started.

Join the Team

Want to shape the future of automation? Check out our job posts and join our team!

What does n8n mean?

Short answer: It means "nodemation" and is pronounced as n-eight-n.

Long answer: "I get that question quite often (more often than I expected) so I decided it is probably best to answer it here. While looking for a good name for the project with a free domain I realized very quickly that all the good ones I could think of were already taken. So, in the end, I chose nodemation. 'node-' in the sense that it uses a Node-View and that it uses Node.js and '-mation' for 'automation' which is what the project is supposed to help with. However, I did not like how long the name was and I could not imagine writing something that long every time in the CLI. That is when I then ended up on 'n8n'." - **Jan Oberhauser, Founder and CEO, n8n.io**

---

cypress/README.md

Debugging Flaky End-to-End Tests - Usage

To debug flaky end-to-end (E2E) tests, use the following command:

```
pnpm run debug:flaky:e2e -- <grep_filter> <burn_count>
```

Parameters:

- `<grep_filter>`: (Optional) A string to filter tests by their `it()` or `describe()` block titles, or by tags if using the `@cypress/grep` plugin. If omitted, all tests will be run.
- `<burn_count>`: (Optional) The number of times to run the filtered tests. Defaults to 5 if not provided.

Examples:

1. **Run all tests tagged with `CAT-726` ten times:**

   ```
   pnpm run debug:flaky:e2e CAT-726 10
   ```

2. **Run all tests containing "login" five times (default burn count):**

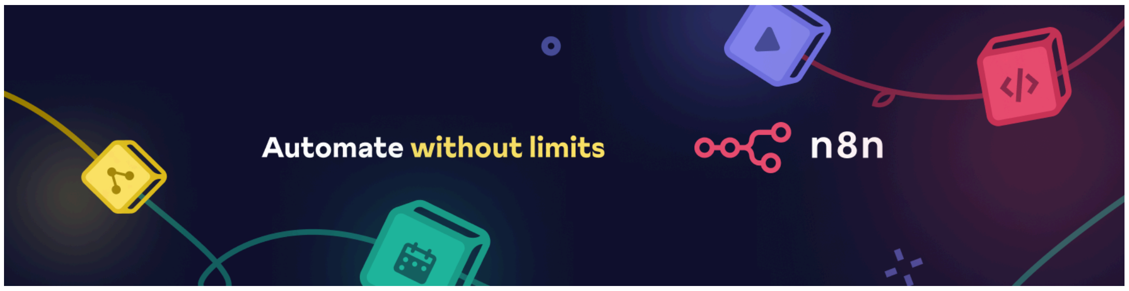
   ```
   pnpm run debug:flaky:e2e login
   ```

3. **Run all tests five times (default grep and burn count):**

```
pnpm run debug:flaky:e2e
```

---

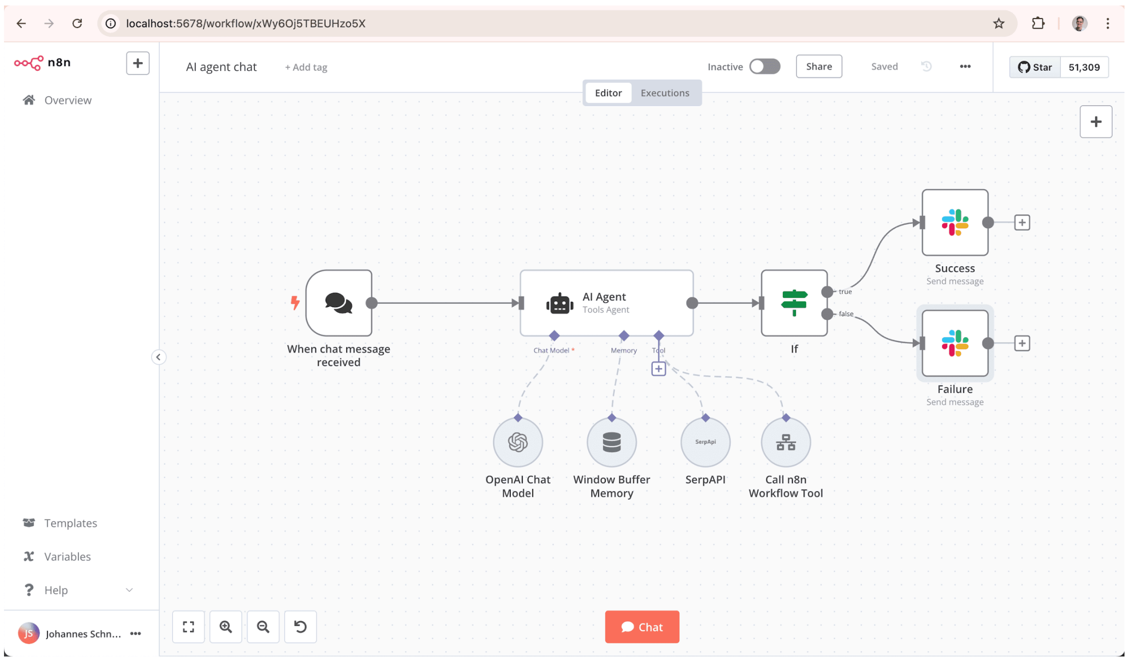
docker/images/n8n/README.md



n8n.io - Workflow Automation

n8n - Secure Workflow Automation for Technical Teams

n8n is a workflow automation platform that gives technical teams the flexibility of code with the speed of no-code. With 400+ integrations, native AI capabilities, and a fair-code license, n8n lets you build powerful automations while maintaining full control over your data and deployments.



n8n.io - Screenshot

Key Capabilities

- **Code When You Need It**: Write JavaScript/Python, add npm packages, or use the visual interface
- **AI-Native Platform**: Build AI agent workflows based on LangChain with your own data and models
- **Full Control**: Self-host with our fair-code license or use our cloud offering
- **Enterprise-Ready**: Advanced permissions, SSO, and air-gapped deployments
- **Active Community**: 400+ integrations and 900+ ready-to-use templates

Contents

Demo

This :tv: short video (< 4 min) goes over key concepts of creating workflows in n8n.

Available integrations

n8n has 200+ different nodes to automate workflows. A full list can be found at https://n8n.io/integrations.

Documentation

The official n8n documentation can be found at https://docs.n8n.io.

Additional information and example workflows are available on the website at https://n8n.io.

Start n8n in Docker

In the terminal, enter the following:

```
docker volume create n8n_data

docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n
```

This command will download the required n8n image and start your container. You can then access n8n by opening: http://localhost:5678

To save your work between container restarts, it also mounts a docker volume, `n8n_data`. The workflow data gets saved in an SQLite database in the user folder (`/home/node/.n8n`). This folder also contains important data like the webhook URL and the encryption key used for securing credentials.

If this data can't be found at startup n8n automatically creates a new key and any existing credentials can no longer be decrypted.

Start n8n with tunnel

> **WARNING**: This is only meant for local development and testing and should **NOT** be used in production!

n8n must be reachable from the internet to make use of webhooks - essential for triggering workflows from external web-based services such as GitHub. To make this easier, n8n has a special tunnel service which redirects requests from our servers to your local n8n instance. You can inspect the code running this service here: https://github.com/n8n-io/localtunnel

To use it simply start n8n with `--tunnel`

```
docker volume create n8n_data
docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n \
 start --tunnel
```

Use with PostgreSQL

By default, n8n uses SQLite to save credentials, past executions and workflows. However, n8n also supports using PostgreSQL.

> **WARNING**: Even when using a different database, it is still important to persist the `/home/node/.n8n` folder, which also contains essential n8n user data including the encryption key for the credentials.

In the following commands, replace the placeholders (depicted within angled brackets, e.g. `<POSTGRES_USER>`) with the actual data:

```
docker volume create n8n_data

docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e DB_TYPE=postgresdb \
 -e DB_POSTGRESDB_DATABASE=<POSTGRES_DATABASE> \
 -e DB_POSTGRESDB_HOST=<POSTGRES_HOST> \
 -e DB_POSTGRESDB_PORT=<POSTGRES_PORT> \
 -e DB_POSTGRESDB_USER=<POSTGRES_USER> \
 -e DB_POSTGRESDB_SCHEMA=<POSTGRES_SCHEMA> \
 -e DB_POSTGRESDB_PASSWORD=<POSTGRES_PASSWORD> \
 -v n8n_data:/home/node/.n8n \
 docker.n8n.io/n8nio/n8n
```

A full working setup with docker-compose can be found here.

Passing sensitive data using files

To avoid passing sensitive information via environment variables, "_FILE" may be appended to some environment variable names. n8n will then load the data from a file with the given name. This makes it possible to load data easily from Docker and Kubernetes secrets.

The following environment variables support file input:

- DB_POSTGRESDB_DATABASE_FILE
- DB_POSTGRESDB_HOST_FILE
- DB_POSTGRESDB_PASSWORD_FILE
- DB_POSTGRESDB_PORT_FILE

- DB_POSTGRESDB_USER_FILE
- DB_POSTGRESDB_SCHEMA_FILE

Example server setups

Example server setups for a range of cloud providers and scenarios can be found in the Server Setup documentation.

Updating

Before you upgrade to the latest version make sure to check here if there are any breaking changes which may affect you: Breaking Changes

From your Docker Desktop, navigate to the Images tab and select Pull from the context menu to download the latest n8n image.

You can also use the command line to pull the latest, or a specific version:

Pull latest (stable) version

```
docker pull docker.n8n.io/n8nio/n8n
```

Pull specific version

```
docker pull docker.n8n.io/n8nio/n8n:0.220.1
```

Pull next (unstable) version

```
docker pull docker.n8n.io/n8nio/n8n:next
```

Stop the container and start it again:

1. Get the container ID:

```
docker ps -a
```

2. Stop the container with ID container_id:

```
docker stop [container_id]
```

3. Remove the container (this does not remove your user data) with ID container_id:

```
docker rm [container_id]
```

4. Start the new container:

```
docker run --name=[container_name] [options] -d docker.n8n.io/n8nio/n8n
```

Updating with Docker Compose

If you run n8n using a Docker Compose file, follow these steps to update n8n:

```
# Pull latest version
docker compose pull
```

```
# Stop and remove older version
docker compose down

# Start the container
docker compose up -d
```

Setting the timezone

To specify the timezone n8n should use, the environment variable `GENERIC_TIMEZONE` can be set. One example where this variable has an effect is the Schedule node.

The system's timezone can be set separately with the environment variable `TZ`. This controls the output of certain scripts and commands such as `$ date`.

For example, to use the same timezone for both:

```
docker run -it --rm \
 --name n8n \
 -p 5678:5678 \
 -e GENERIC_TIMEZONE="Europe/Berlin" \
 -e TZ="Europe/Berlin" \
 docker.n8n.io/n8nio/n8n
```

For more information on configuration and environment variables, please see the n8n documentation.

Build Docker-Image

```
docker buildx build --platform linux/amd64,linux/arm64 --build-arg
N8N_VERSION=<VERSION> -t n8n:<VERSION> .

# For example:
docker buildx build --platform linux/amd64,linux/arm64 --build-arg
N8N_VERSION=1.30.1 -t n8n:1.30.1 .
```

What does n8n mean and how do you pronounce it?

Short answer: It means "nodemation" and it is pronounced as n-eight-n.

Long answer: I get that question quite often (more often than I expected) so I decided it is probably best to answer it here. While looking for a good name for the project with a free domain I realized very quickly that all the good ones I could think of were already taken. So, in the end, I chose nodemation. "node-" in the sense that it uses a Node-View and that it uses Node.js and "-mation" for "automation" which is what the project is supposed to help with. However, I did not like how long the name was and I could not imagine writing something that long every time in the CLI. That is when I then ended up on "n8n". Sure it does not work perfectly but neither does it for Kubernetes (k8s) and I did not hear anybody complain there. So I guess it should be ok.

Support

If you need more help with n8n, you can ask for support in the n8n community forum. This is the best source of answers, as both the n8n support team and community members can help.

Jobs

If you are interested in working for n8n and so shape the future of the project check out our job posts.

License

You can find the license information here.

---

packages/@n8n/api-types/README.md

@n8n/api-types

This package contains types and schema definitions for the n8n internal API, so that these can be shared between the backend and the frontend code.

---

packages/@n8n/benchmark/README.md

n8n benchmarking tool

Tool for executing benchmarks against an n8n instance.

Directory structure

```
packages/@n8n/benchmark
├── scenarios          Benchmark scenarios
├── src                Source code for the n8n-benchmark cli
├── Dockerfile         Dockerfile for the n8n-benchmark cli
├── scripts            Orchestration scripts
```

Benchmarking an existing n8n instance

The easiest way to run the existing benchmark scenarios is to use the benchmark docker image:

```
docker pull ghcr.io/n8n-io/n8n-benchmark:latest
# Print the help to list all available flags
docker run ghcr.io/n8n-io/n8n-benchmark:latest run --help
# Run all available benchmark scenarios for 1 minute with 5 concurrent
```

```
requests
docker run ghcr.io/n8n-io/n8n-benchmark:latest run \
    --n8nBaseUrl=https://instance.url \
    --n8nUserEmail=InstanceOwner@email.com \
    --n8nUserPassword=InstanceOwnerPassword \
    --vus=5 \
    --duration=1m \
    --scenarioFilter=single-webhook
```

Using custom scenarios with the Docker image

It is also possible to create your own benchmark scenarios and load them using the `--testScenariosPath` flag:

```
# Assuming your scenarios are located in `./scenarios`, mount them into
`/scenarios` in the container
docker run -v ./scenarios:/scenarios ghcr.io/n8n-io/n8n-benchmark:latest run
\
    --n8nBaseUrl=https://instance.url \
    --n8nUserEmail=InstanceOwner@email.com \
    --n8nUserPassword=InstanceOwnerPassword \
    --vus=5 \
    --duration=1m \
    --testScenariosPath=/scenarios
```

Running the entire benchmark suite

The benchmark suite consists of benchmark scenarios and different n8n setups.

locally

```
pnpm benchmark-locally
```

In the cloud

```
pnpm benchmark-in-cloud
```

Running the `n8n-benchmark` cli

The `n8n-benchmark` cli is a node.js program that runs one or more scenarios against a single n8n instance.

Locally with Docker

Build the Docker image:

```
# Must be run in the repository root
# k6 doesn't have an arm64 build available for linux, we need to build
against amd64
docker build --platform linux/amd64 -t n8n-benchmark -f
packages/@n8n/benchmark/Dockerfile .
```

Run the image

```
docker run \
  -e N8N_USER_EMAIL=user@n8n.io \
  -e N8N_USER_PASSWORD=password \
  # For macos, n8n running outside docker
  -e N8N_BASE_URL=http://host.docker.internal:5678 \
  n8n-benchmark
```

Locally without Docker

Requirements:

- k6
- Node.js v20 or higher

```
pnpm build

# Run tests against http://localhost:5678 with specified email and password
N8N_USER_EMAIL=user@n8n.io N8N_USER_PASSWORD=password ./bin/n8n-benchmark run
```

Benchmark scenarios

A benchmark scenario defines one or multiple steps to execute and measure. It consists of:

- Manifest file which describes and configures the scenario
- Any test data that is imported before the scenario is run
- A `k6` script which executes the steps and receives `API_BASE_URL` environment variable in runtime.

Available scenarios are located in `./scenarios`.

n8n setups

A n8n setup defines a single n8n runtime configuration using Docker compose. Different n8n setups are located in `./scripts/n8nSetups`.

---

packages/@n8n/codemirror-lang/README.md

@n8n/codemirror-lang

Language support package for CodeMirror 6 in n8n

n8n Expression Language support

---

packages/@n8n/codemirror-lang/src/expressions/README.md

n8n Expression language support

Usage

```
import { parserWithMetaData as n8nParser } from '@n8n/codemirror-lang';
import { LanguageSupport, LRLanguage } from '@codemirror/language';
import { parseMixed } from '@lezer/common';
import { parser as jsParser } from '@lezer/javascript';

const n8nPlusJsParser = n8nParser.configure({
    wrap: parseMixed((node) => {
        if (node.type.isTop) return null;

        return node.name === 'Resolvable'
            ? { parser: jsParser, overlay: (node) => node.type.name ===
'Resolvable' }
            : null;
    }),
});

const n8nLanguage = LRLanguage.define({ parser: n8nPlusJsParser });

export function n8nExpressionLanguageSupport() {
    return new LanguageSupport(n8nLanguage);
}
```

Supported Unicode ranges

- From `Basic Latin` up to and including `Currency Symbols`
- `Miscellaneous Symbols and Pictographs`
- `CJK Unified Ideographs`

---

packages/@n8n/di/README.md

@n8n/di

`@n8n/di` is a dependency injection (DI) container library, based on `typedi`.

n8n no longer uses `typedi` because:

- `typedi` is no longer officially maintained
- Need for future-proofing, e.g. stage-3 decorators
- Small enough that it is worth the maintenance burden
- Easier to customize, e.g. to simplify unit tests

Usage

```
// from https://github.com/typestack/typedi/blob/develop/README.md
import { Container, Service } from 'typedi';

@Service()
class ExampleInjectedService {
  printMessage() {
    console.log('I am alive!');
  }
}

@Service()
class ExampleService {
  constructor(
    // because we annotated ExampleInjectedService with the @Service()
    // decorator TypeDI will automatically inject an instance of
    // ExampleInjectedService here when the ExampleService class is requested
    // from TypeDI.
    public injectedService: ExampleInjectedService
  ) {}
}

const serviceInstance = Container.get(ExampleService);
// we request an instance of ExampleService from TypeDI

serviceInstance.injectedService.printMessage();
// logs "I am alive!" to the console
```

Requires enabling these flags in `tsconfig.json`:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

---

packages/@n8n/extension-sdk/README.md

@n8n/plugin-sdk

---

packages/@n8n/json-schema-to-zod/README.md

Json-Schema-to-Zod

A package to convert JSON schema (draft 4+) objects into Zod schemas in the form of Zod objects at runtime.

Installation

```
npm install @n8n/json-schema-to-zod
```

Simple example

```
import { jsonSchemaToZod } from "json-schema-to-zod";

const jsonSchema = {
  type: "object",
  properties: {
    hello: {
      type: "string",
    },
  },
};

const zodSchema = jsonSchemaToZod(myObject);
```

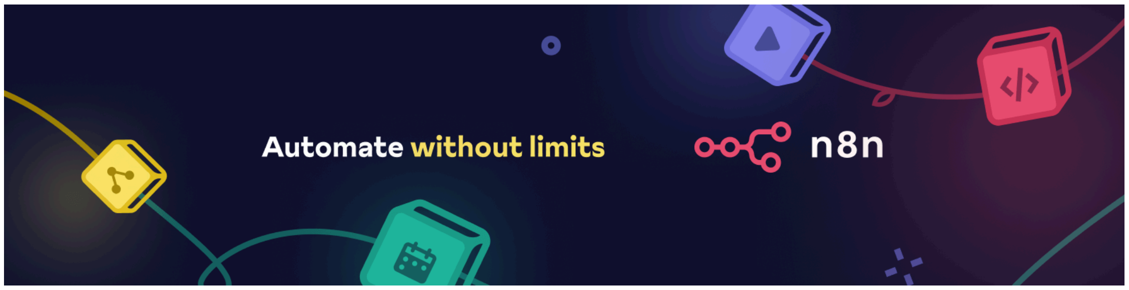
Overriding a parser

You can pass a function to the `overrideParser` option, which represents a function that receives the current schema node and the reference object, and should return a zod object when it wants to replace a default output. If the default output should be used for the node just return undefined.

Acknowledgements

This is a fork of `json-schema-to-zod`.

packages/@n8n/nodes-langchain/README.md



Banner image

n8n-nodes-langchain

This repo contains nodes to use n8n in combination with LangChain.

These nodes are still in Beta state and are only compatible with the Docker image `docker.n8n.io/n8nio/n8n:ai-beta`.

License

You can find the license information here

---

packages/@n8n/nodes-langchain/nodes/vector_store/shared/createVectorStoreNode/README.md

Overview

`createVectorStoreNode` is a factory function that generates n8n nodes for vector store operations. It abstracts the common functionality needed for vector stores while allowing specific implementations to focus only on their unique aspects.

Purpose

The function provides a standardized way to: 1. Create vector store nodes with consistent UIs 2. Handle different operation modes (load, insert, retrieve, update, retrieve-as-tool) 3. Process documents and embeddings 4. Maintain connection to LLM services

Architecture

```
    /createVectorStoreNode/                        # Create Vector Store Node
    /constants.ts                        # Constants like operation modes and
descriptions
    /types.ts                            # TypeScript interfaces and types
    /utils.ts                        # Utility functions for node
configuration
    /createVectorStoreNode.ts        # Main factory function
    /processDocuments.ts             # Document processing helpers
    /operations/                     # Operation-specific logic
      /loadOperation.ts              # Handles 'load' mode
      /insertOperation.ts            # Handles 'insert' mode
      /updateOperation.ts            # Handles 'update' mode
      /retrieveOperation.ts          # Handles 'retrieve' mode
      /retrieveAsToolOperation.ts    # Handles 'retrieve-as-tool' mode
```

Usage

To create a new vector store node:

```
import { createVectorStoreNode } from './createVectorStoreNode';

export class MyVectorStoreNode {
  static description = createVectorStoreNode({
    meta: {
      displayName: 'My Vector Store',
      name: 'myVectorStore',
      description: 'Operations for My Vector Store',
      docsUrl: 'https://docs.example.com/my-vector-store',
      icon: 'file:myIcon.svg',
      // Optional: specify which operations this vector store supports
      operationModes: ['load', 'insert', 'update','retrieve',
'retrieve-as-tool'],
    },
    sharedFields: [
      // Fields shown in all operation modes
    ],
    loadFields: [
      // Fields specific to 'load' operation
    ],
    insertFields: [
      // Fields specific to 'insert' operation
    ],
    retrieveFields: [
      // Fields specific to 'retrieve' operation
    ],
    // Functions to implement
    getVectorStoreClient: async (context, filter, embeddings, itemIndex) => {
      // Create and return vector store instance
    },
    populateVectorStore: async (context, embeddings, documents, itemIndex) =>
{
      // Insert documents into vector store
    },
    // Optional: cleanup function - called in finally blocks after operations
    releaseVectorStoreClient: (vectorStore) => {
      // Release resources such as database connections or external clients
      // For example, in PGVector: vectorStore.client?.release();
    },
  });
}
```

Operation Modes

1. `load` Mode

- Retrieves documents from the vector store based on a query
- Embeds the query and performs similarity search
- Returns ranked documents with their similarity scores

2. `insert` Mode

- Processes documents from input
- Embeds and stores documents in the vector store
- Returns serialized documents with metadata
- Supports batched processing with configurable embedding batch size

3. `retrieve` Mode

- Returns the vector store instance for use with AI nodes
- Allows LLMs to query the vector store directly
- Used with chains and retrievers

4. `retrieve-as-tool` Mode

- Creates a tool that wraps the vector store
- Allows AI agents to use the vector store as a tool
- Returns documents in a format digestible by agents

5. `update` Mode (optional)

- Updates existing documents in the vector store by ID
- Requires the vector store to support document updates
- Only enabled if included in `operationModes`
- Uses `addDocuments` method with an `ids` array to update specific documents
- Processes a single document per item and applies it to the specified ID
- Validates that only one document is being updated per operation

Key Components

1. NodeConstructorArgs Interface

Defines the configuration and callbacks that specific vector store implementations must provide:

> **Note:** In node version 1.1+, the `populateVectorStore` function must handle receiving multiple documents at once for batch processing.

```
interface VectorStoreNodeConstructorArgs<T extends VectorStore> {
  meta: NodeMeta;                    // Node metadata (name, description, 
etc.)
  methods?: { ... };                 // Optional methods for list searches
  sharedFields: INodeProperties[];   // Fields shown in all modes
  insertFields?: INodeProperties[];  // Fields specific to insert mode
  loadFields?: INodeProperties[];    // Fields specific to load mode
  retrieveFields?: INodeProperties[]; // Fields specific to retrieve mode
  updateFields?: INodeProperties[];  // Fields specific to update mode

  // Core implementation functions
  populateVectorStore: Function;     // Store documents in vector store 
(accepts batches in v1.1+)
  getVectorStoreClient: Function;    // Get vector store instance
  releaseVectorStoreClient?: Function; // Clean up resources
}
```

2. Operation Handlers

Each operation mode has its own handler module with a well-defined interface:

```
// Example: loadOperation.ts
export async function handleLoadOperation<T extends VectorStore>(
  context: IExecuteFunctions,
  args: VectorStoreNodeConstructorArgs<T>,
  embeddings: Embeddings,
  itemIndex: number
): Promise<INodeExecutionData[]>

// Example: insertOperation.ts (v1.1+)
export async function handleInsertOperation<T extends VectorStore>(
  context: IExecuteFunctions,
  args: VectorStoreNodeConstructorArgs<T>,
  embeddings: Embeddings
): Promise<INodeExecutionData[]>
```

3. Document Processing

The `processDocument` function standardizes how documents are handled:

```
const { processedDocuments, serializedDocuments } = await processDocument(
  documentInput,
  itemData,
  itemIndex
);
```

Implementation Details

Error Handling and Resource Management

Each operation handler includes error handling with proper resource cleanup. The `releaseVectorStoreClient` function is called in a `finally` block to ensure resources are released even if an error occurs:

```
try {
  // Operation logic
} finally {
  // Release resources even if an error occurs
  args.releaseVectorStoreClient?.(vectorStore);
}
```

When releaseVectorStoreClient is called:

- After completing a similarity search in `loadOperation`
- As part of the `closeFunction` in `retrieveOperation` to release resources when they're no longer needed
- After each tool use in `retrieveAsToolOperation`
- After updating documents in `updateOperation`
- After inserting documents in `insertOperation`

This design ensures proper resource management, which is especially important for database-backed vector stores (like PGVector) that need to return connections to a pool. Without proper cleanup, prolonged usage could lead to resource leaks or connection pool exhaustion.

Dynamic Tool Creation

For the `retrieve-as-tool` mode, a DynamicTool is created that exposes vector store functionality:

```
const vectorStoreTool = new DynamicTool({
  name: toolName,
  description: toolDescription,
  func: async (input) => {
    // Search vector store with input
    // ...
  },
});
```

Performance Considerations

1. **Resource Management**: Each operation properly handles resource cleanup with `releaseVectorStoreClient`.

2. **Batched Processing**: The `insert` operation processes documents in configurable batches. In node version 1.1+, a single embedding operation is performed for all documents in a batch, significantly improving performance by reducing API calls.

3. **Metadata Filtering**: Filters can be applied during search operations to reduce result sets.

4. **Execution Cancellation**: The code checks for cancellation signals to stop processing when needed.

---

packages/@n8n/utils/README.md

@n8n/utils

A collection of utility functions that provide common functionality for both Front-End and Back-End packages.

Table of Contents

Features

- **Reusable Logic**: Build complex, stateful functionality using modular composable functions that you can easily reuse.
- **Consistent Patterns**: Enjoy a unified approach across n8n packages, making integration and maintenance a breeze.
- **Type-Safe & Reliable**: Benefit from TypeScript support, which improves the developer experience and code robustness.
- **Universal Functionality**: Designed to work seamlessly on both the front-end and back-end.
- **Easily Testable**: A modular design that simplifies testing, maintenance, and rapid development.

Contributing

For more details, please read our CONTRIBUTING.md.

License

For more details, please read our LICENSE.md.

---

packages/@n8n/utils/src/search/snapshots/README.md

Search snapshots

This directory contains snapshots containing real data fed into the sublimeSearch function.

These were obtained via `console.log(items)` right before the sublimeSearch call in `editor-ui` (currently in `packages/frontend/editor-ui/src/components/Node/NodeCreator/utils.ts`) Which is triggered by typing in the search bar in varying states of the application:

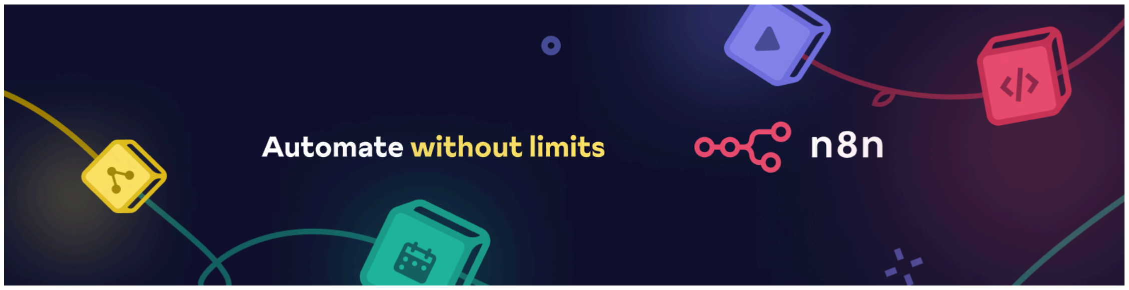
- toplevel: From an empty workflow (so missing e.g. tools)

After typing in the search bar you should see an object in the console you can copy via `Right Click->Copy Object” which will cleanly paste to json.

Please use Chrome for capturing these - the recovered object in Chrome is about 3x larger than in Firefox due to Firefox dropping some nested values

---

packages/core/README.md



n8n.io - Workflow Automation

n8n-core

Core components for n8n

```
npm install n8n-core
```

License

You can find the license information here

---

packages/extensions/insights/README.md

@n8n/n8n-extension-insights

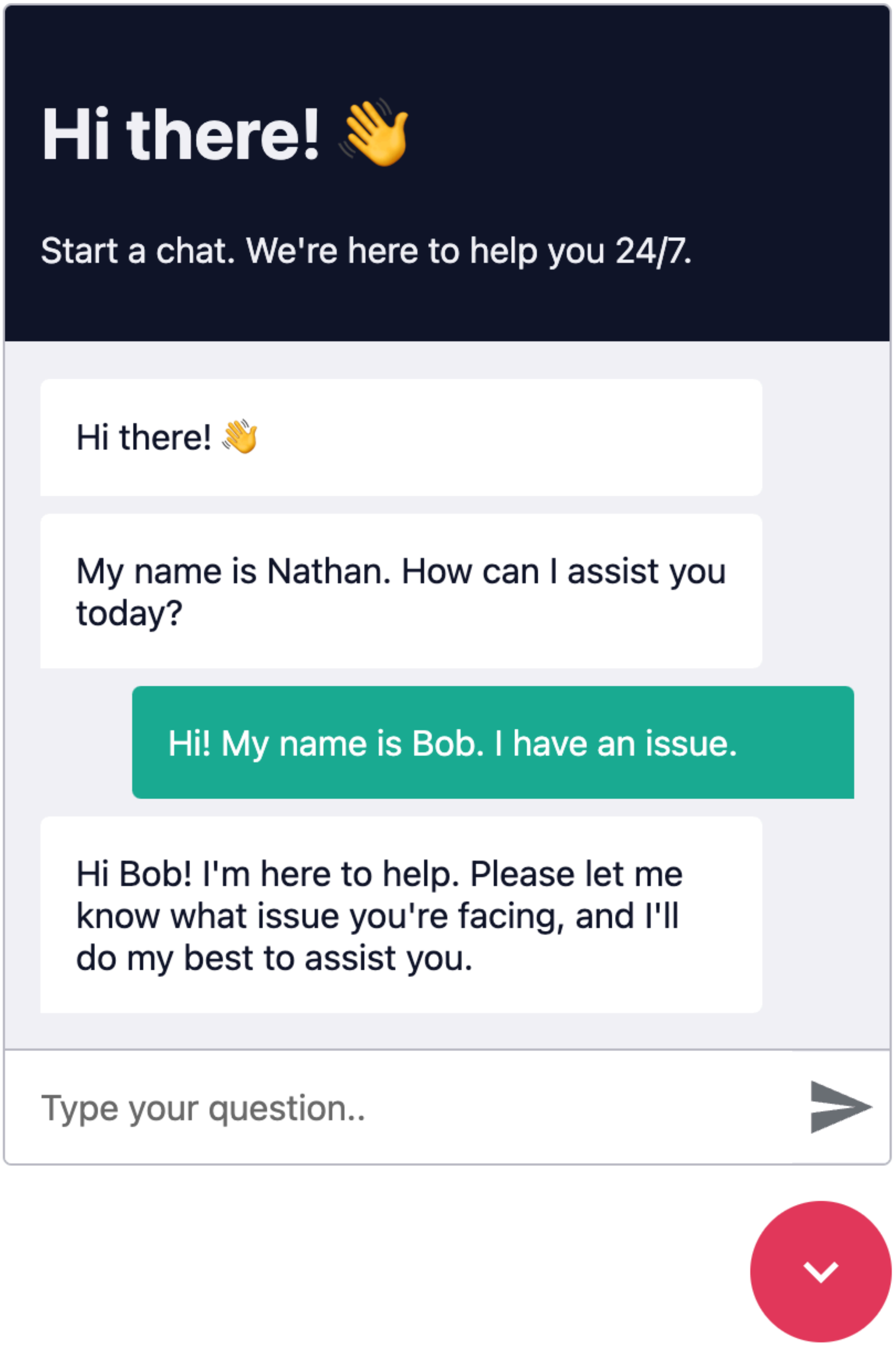
---

packages/frontend/@n8n/chat/README.md

n8n Chat

This is an embeddable Chat widget for n8n. It allows the execution of AI-Powered Workflows through a Chat window.

Windowed Example



Hi there! 👋

Start a chat. We're here to help you 24/7.

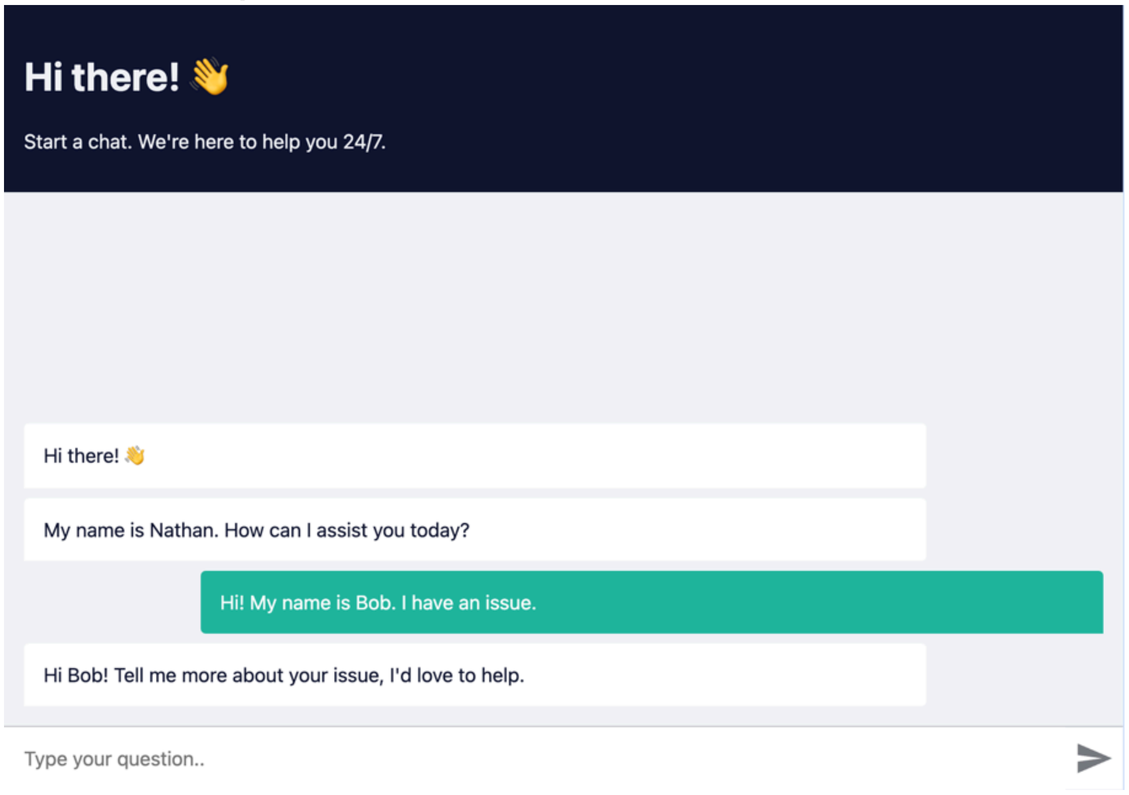
Hi there! 👋

My name is Nathan. How can I assist you today?

Hi! My name is Bob. I have an issue.

Hi Bob! I'm here to help. Please let me know what issue you're facing, and I'll do my best to assist you.

Fullscreen Example


Hi there! 👋

Start a chat. We're here to help you 24/7.

Hi there! 👋

My name is Nathan. How can I assist you today?

Hi! My name is Bob. I have an issue.

Hi Bob! Tell me more about your issue, I'd love to help.

Prerequisites

Create a n8n workflow which you want to execute via chat. The workflow has to be triggered using a **Chat Trigger** node.

Open the **Chat Trigger** node and add your domain to the **Allowed Origins (CORS)** field. This makes sure that only requests from your domain are accepted.

See example workflow

Make sure the workflow is **Active.**

How it works

Each Chat request is sent to the n8n Webhook endpoint, which then sends back a response.

Each request is accompanied by an `action` query parameter, where `action` can be one of: - `loadPreviousSession` - When the user opens the Chatbot again and the previous chat session should be loaded - `sendMessage` - When the user sends a message

Installation

Open the **Webhook** node and replace `YOUR_PRODUCTION_WEBHOOK_URL` with your production URL. This is the URL that the Chat widget will use to send requests to.

a. CDN Embed

Add the following code to your HTML page.

```
<link href="https://cdn.jsdelivr.net/npm/@n8n/chat/dist/style.css"
rel="stylesheet" />
<script type="module">
    import { createChat } from
'https://cdn.jsdelivr.net/npm/@n8n/chat/dist/chat.bundle.es.js';

    createChat({
        webhookUrl: 'YOUR_PRODUCTION_WEBHOOK_URL'
    });
</script>
```

b. Import Embed

Install and save n8n Chat as a production dependency.

```
npm install @n8n/chat
```

Import the CSS and use the `createChat` function to initialize your Chat window.

```
import '@n8n/chat/style.css';
import { createChat } from '@n8n/chat';

createChat({
    webhookUrl: 'YOUR_PRODUCTION_WEBHOOK_URL'
});
```

Vue.js

```
<script lang="ts" setup>
// App.vue
import { onMounted } from 'vue';
import '@n8n/chat/style.css';
import { createChat } from '@n8n/chat';

onMounted(() => {
    createChat({
        webhookUrl: 'YOUR_PRODUCTION_WEBHOOK_URL'
    });
});
</script>
<template>
    <div></div>
</template>
```

React

```
// App.tsx
import { useEffect } from 'react';
import '@n8n/chat/style.css';
import { createChat } from '@n8n/chat';

export const App = () => {
    useEffect(() => {
        createChat({
            webhookUrl: 'YOUR_PRODUCTION_WEBHOOK_URL'
        });
    }, []);

    return (<div></div>);
};
```

Options

The default options are:

```
createChat({
    webhookUrl: '',
    webhookConfig: {
        method: 'POST',
        headers: {}
    },
    target: '#n8n-chat',
    mode: 'window',
    chatInputKey: 'chatInput',
    chatSessionKey: 'sessionId',
    loadPreviousSession: true,
    metadata: {},
    showWelcomeScreen: false,
    defaultLanguage: 'en',
    initialMessages: [
        'Hi there! 👋',
        'My name is Nathan. How can I assist you today?'
    ],
    i18n: {
        en: {
            title: 'Hi there! 👋',
            subtitle: "Start a chat. We're here to help you 24/7.",
            footer: '',
            getStarted: 'New Conversation',
            inputPlaceholder: 'Type your question..',
        },
    },
});
```

`webhookUrl`

- **Type**: `string`
- **Required**: `true`

- **Examples**:
  - `https://yourname.app.n8n.cloud/webhook/513107b3-6f3a-4a1e-af21-659f0ed14183`
  - `http://localhost:5678/webhook/513107b3-6f3a-4a1e-af21-659f0ed14183`
- **Description**: The URL of the n8n Webhook endpoint. Should be the production URL.

`webhookConfig`

- **Type**: `{ method: string, headers: Record<string, string> }`
- **Default**: `{ method: 'POST', headers: {} }`
- **Description**: The configuration for the Webhook request.

`target`

- **Type**: `string`
- **Default**: `'#n8n-chat'`
- **Description**: The CSS selector of the element where the Chat window should be embedded.

`mode`

- **Type**: `'window' | 'fullscreen'`
- **Default**: `'window'`
- **Description**: The render mode of the Chat window.
  - In `window` mode, the Chat window will be embedded in the target element as a chat toggle button and a fixed size chat window.
  - In `fullscreen` mode, the Chat will take up the entire width and height of its target container.

`showWelcomeScreen`

- **Type**: `boolean`
- **Default**: `false`
- **Description**: Whether to show the welcome screen when the Chat window is opened.

`chatInputKey`

- **Type**: `string`
- **Default**: `'chatInput'`
- **Description**: The key to use for sending the chat input for the AI Agent node.

`chatSessionKey`

- **Type**: `string`
- **Default**: `'sessionId'`
- **Description**: The key to use for sending the chat history session ID for the AI Memory node.

`loadPreviousSession`

- **Type**: `boolean`
- **Default**: `true`
- **Description**: Whether to load previous messages (chat context).

`defaultLanguage`

- **Type**: `string`
- **Default**: `'en'`
- **Description**: The default language of the Chat window. Currently only `en` is supported.

`i18n`

- **Type**: `{ [key: string]: Record<string, string> }`
- **Description**: The i18n configuration for the Chat window. Currently only `en` is supported.

`initialMessages`

- **Type**: `string[]`
- **Description**: The initial messages to be displayed in the Chat window.

`allowFileUploads`

- **Type**: `Ref<boolean> | boolean`
- **Default**: `false`
- **Description**: Whether to allow file uploads in the chat. If set to `true`, users will be able to upload files through the chat interface.

`allowedFilesMimeTypes`

- **Type**: `Ref<string> | string`
- **Default**: `''`
- **Description**: A comma-separated list of allowed MIME types for file uploads. Only applicable if `allowFileUploads` is set to `true`. If left empty, all file types are allowed. For example: `'image/*,application/pdf'`.

Customization

The Chat window is entirely customizable using CSS variables.

```css
:root {
    --chat--color-primary: #e74266;
    --chat--color-primary-shade-50: #db4061;
    --chat--color-primary-shade-100: #cf3c5c;
    --chat--color-secondary: #20b69e;
    --chat--color-secondary-shade-50: #1ca08a;
    --chat--color-white: #ffffff;
    --chat--color-light: #f2f4f8;
    --chat--color-light-shade-50: #e6e9f1;
    --chat--color-light-shade-100: #c2c5cc;
    --chat--color-medium: #d2d4d9;
    --chat--color-dark: #101330;
    --chat--color-disabled: #777980;
    --chat--color-typing: #404040;

    --chat--spacing: 1rem;
    --chat--border-radius: 0.25rem;
    --chat--transition-duration: 0.15s;

    --chat--window--width: 400px;
    --chat--window--height: 600px;

    --chat--header-height: auto;
    --chat--header--padding: var(--chat--spacing);
    --chat--header--background: var(--chat--color-dark);
    --chat--header--color: var(--chat--color-light);
    --chat--header--border-top: none;
    --chat--header--border-bottom: none;
    --chat--header--border-bottom: none;
    --chat--header--border-bottom: none;
    --chat--heading--font-size: 2em;
    --chat--header--color: var(--chat--color-light);
    --chat--subtitle--font-size: inherit;
    --chat--subtitle--line-height: 1.8;

    --chat--textarea--height: 50px;

    --chat--message--font-size: 1rem;
    --chat--message--padding: var(--chat--spacing);
    --chat--message--border-radius: var(--chat--border-radius);
    --chat--message-line-height: 1.8;
    --chat--message--bot--background: var(--chat--color-white);
    --chat--message--bot--color: var(--chat--color-dark);
    --chat--message--bot--border: none;
    --chat--message--user--background: var(--chat--color-secondary);
    --chat--message--user--color: var(--chat--color-white);
    --chat--message--user--border: none;
    --chat--message--pre--background: rgba(0, 0, 0, 0.05);

    --chat--toggle--background: var(--chat--color-primary);
    --chat--toggle--hover--background: var(--chat--color-primary-shade-50);
    --chat--toggle--active--background: var(--chat--color-primary-shade-100);
    --chat--toggle--color: var(--chat--color-white);
    --chat--toggle--size: 64px;
}
```

Caveats

Fullscreen mode

In fullscreen mode, the Chat window will take up the entire width and height of its target container. Make sure that the container has a set width and height.

```
html,
body,
#n8n-chat {
    width: 100%;
    height: 100%;
}
```

License

You can find the license information here

---

packages/frontend/@n8n/composables/README.md

@n8n/composables

A collection of Vue composables that provide common functionality across n8n's Front-End packages.

Table of Contents

Features

- **Reusable Logic**: Encapsulate complex stateful logic into composable functions.
- **Consistency**: Ensure consistent patterns and practices across our Vue components.
- **Extensible**: Easily add new composables as our project grows.
- **Optimized**: Fully compatible with the Composition API.

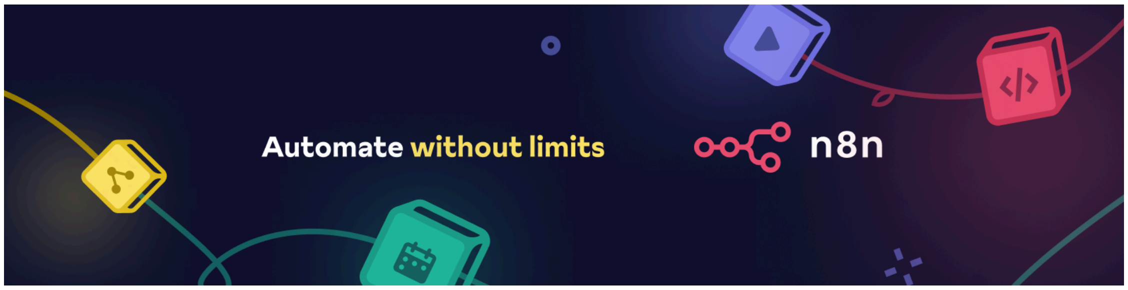
Contributing

For more details, please read our CONTRIBUTING.md.

License

For more details, please read our LICENSE.md.

packages/frontend/@n8n/design-system/README.md



n8n.io - Workflow Automation

@n8n/design-system

A component system for n8n using Storybook to preview.

Project setup

```
pnpm install
```

Compiles and hot-reloads for development

```
pnpm storybook
```

Build static pages

```
pnpm build:storybook
```

Run your unit tests

```
pnpm test:unit
```

Lints and fixes files

```
pnpm lint
```

Build css files

```
pnpm build:theme
```

Monitor theme files and build any changes

```
pnpm watch:theme
```

License

You can find the license information here

packages/frontend/@n8n/stores/README.md

@n8n/stores

A collection of Pinia stores that provide common data-related functionality across n8n's Front-End packages.

Table of Contents

Features

- **Composable State Management**: Share and reuse stateful logic across multiple Vue components using Pinia stores.
- **Consistent Patterns**: Promote uniform state handling and best practices throughout the front-end codebase.
- **Easy Extensibility**: Add or modify stores as project requirements evolve, supporting scalable development.
- **Composition API Support**: Designed to work seamlessly with Vue's Composition API for modern, maintainable code.

Contributing

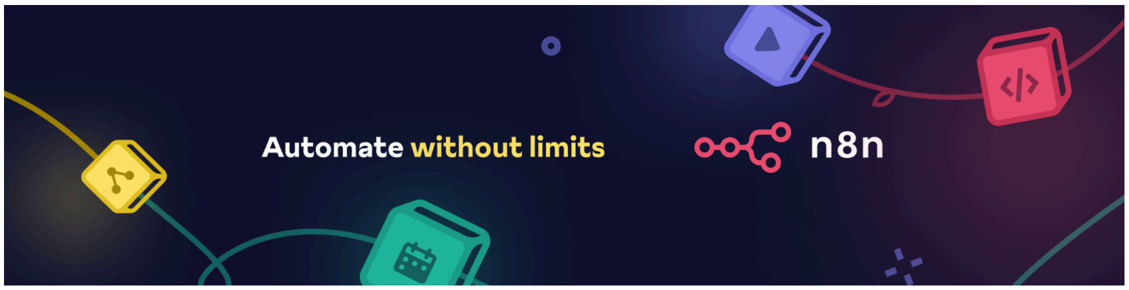
For more details, please read our CONTRIBUTING.md.

License

For more details, please read our LICENSE.md.

---

packages/frontend/editor-ui/README.md



n8n.io - Workflow Automation

n8n-editor-ui

The UI to create and update n8n workflows

```
npm install n8n -g
```

Project setup

```
pnpm install
```

Compiles and hot-reloads for development

```
pnpm serve
```

Compiles and minifies for production

```
pnpm build
```

Run your tests

```
pnpm test
```

Lints and fixes files

```
pnpm lint
```

Run your end-to-end tests

```
pnpm test:e2e
```

Run your unit tests

```
pnpm test:unit
```

Customize configuration

See Configuration Reference.

License

You can find the license information here

---

packages/frontend/editor-ui/src/plugins/i18n/docs/README.md

i18n in n8n

Scope

n8n allows for internalization of the majority of UI text:

- base text, e.g. menu display items in the left-hand sidebar menu,
- node text, e.g. parameter display names and placeholders in the node view,
- credential text, e.g. parameter display names and placeholders in the credential modal,
- header text, e.g. node display names and descriptions at various spots.

Currently, n8n does *not* allow for internalization of:

- messages from outside the `editor-ui` package, e.g. `No active database connection`,
- strings in certain Vue components, e.g. date time picker
- node subtitles, e.g. `create: user` or `getAll: post` below the node name on the canvas,
- new version notification contents in the updates panel, e.g. `Includes node enhancements`, and
- options that rely on `loadOptionsMethod`.

Pending functionality:

- Search in nodes panel by translated node name
- UI responsiveness to differently sized strings
- Locale-aware number formatting

Locale identifiers

A **locale identifier** is a language code compatible with the `Accept-Language` header, e.g. `de` (German), `es` (Spanish), `ja` (Japanese). Regional variants of locale identifiers, such as `-AT` in `de-AT`, are *not* supported. For a list of all locale identifiers, see column 639-1 in this table.

By default, n8n runs in the `en` (English) locale. To have run it in a different locale, set the `N8N_DEFAULT_LOCALE` environment variable to a locale identifier. When running in a non-`en` locale, n8n will display UI strings for the selected locale and fall back to `en` for any untranslated strings.

```
export N8N_DEFAULT_LOCALE=de
pnpm start
```

Output:

```
Initializing n8n process
n8n ready on 0.0.0.0, port 5678
Version: 0.156.0
Locale: de

Editor is now accessible via:
http://localhost:5678/
```

```
Press "o" to open in Browser.
```

Base text

Base text is rendered with no dependencies, i.e. base text is fixed and does not change in any circumstances. Base text is supplied by the user in one file per locale in the `/frontend/editor-ui` package.

Locating base text

The base text file for each locale is located at `/packages/frontend/editor-ui/src/plugins/i18n/locales/` and is named `{localeIdentifier}.json`. Keys in the base text file can be Vue component dirs, Vue component names, and references to symbols in those Vue components. These keys are added by the team as the UI is modified or expanded.

```
{
    "nodeCreator.categoryNames.analytics": "🇩🇪 Analytics",
    "nodeCreator.categoryNames.communication": "🇩🇪 Communication",
    "nodeCreator.categoryNames.coreNodes": "🇩🇪 Core Nodes"
}
```

Translating base text

1. Select a new locale identifier, e.g. `de`, copy the `en` JSON base text file with a new name:

```
cp ./packages/frontend/editor-ui/src/plugins/i18n/locales/en.json
./packages/frontend/editor-ui/src/plugins/i18n/locales/de.json
```

2. Find in the UI a string to translate, and search for it in the newly created base text file. Alternatively, find in `/frontend/editor-ui` a call to `i18n.baseText(key)`, e.g. `i18n.baseText('workflowActivator.deactivateWorkflow')`, and take note of the key and find it in the newly created base text file.

   Note: If you cannot find a string in the new base text file, either it does not belong to base text (i.e., the string might be part of header text, credential text, or node text), or the string might belong to the backend, where i18n is currently unsupported.

3. Translate the string value - do not change the key. In the examples below, a string starting with 🇩🇪 stands for a string translated from English into German.

As an optional final step, remove any untranslated strings from the new base text file. Untranslated strings in the new base text file will trigger a fallback to the `en` base text file.

> For information about **interpolation** and **reusable base text**, refer to the Addendum.

Dynamic text

Dynamic text relies on data specific to each node and credential:

- `headerText` and `nodeText` in the **node translation file**
- `credText` in the **credential translation file**

Locating dynamic text

Locating the credential translation file

A credential translation file is placed at
`/nodes-base/credentials/translations/{localeIdentifier}`

```
credentials
    └── translations
        └── de
            ├── githubApi.json
            └── githubOAuth2Api.json
```

Every credential must have its own credential translation file.

The name of the credential translation file must be sourced from the credential's `description.name` property:

```
export class GithubApi implements ICredentialType {
    name = 'githubApi'; // to use for credential translation file
    displayName = 'Github API';
    documentationUrl = 'github';
    properties: INodeProperties[] = [
```

Locating the node translation file

A node translation file is placed at
`/nodes-base/nodes/{node}/translations/{localeIdentifier}`

```
GitHub
    ├── GitHub.node.ts
    ├── GitHubTrigger.node.ts
    └── translations
        └── de
            ├── github.json
            └── githubTrigger.json
```

Every node must have its own node translation file.

> For information about nodes in **versioned dirs** and **grouping dirs**, refer to the Addendum.

The name of the node translation file must be sourced from the node's `description.name` property:

```
export class Github implements INodeType {
    description: INodeTypeDescription = {
        displayName: 'GitHub',
        name: 'github', // to use for node translation file name
        icon: 'file:github.svg',
        group: ['input'],
```

Translating dynamic text

Translating the credential translation file

> **Note**: All translation keys are optional. Missing translation values trigger a fallback to the `en` locale strings.

A credential translation file, e.g. `githubApi.json` is an object containing keys that match the credential parameter names:

```
export class GithubApi implements ICredentialType {
    name = 'githubApi';
    displayName = 'Github API';
    documentationUrl = 'github';
    properties: INodeProperties[] = [
        {
            displayName: 'Github Server',
            name: 'server', // key to use in translation
            type: 'string',
            default: 'https://api.github.com',
            description: 'The server to connect to. Only has to be set if
Github Enterprise is used.',
        },
        {
            displayName: 'User',
            name: 'user', // key to use in translation
            type: 'string',
            default: '',
        },
        {
            displayName: 'Access Token',
            name: 'accessToken', // key to use in translation
            type: 'string',
            default: '',
        },
    ];
}
```

The object for each node credential parameter allows for the keys `displayName`, `description`, and `placeholder`.

```
{
    "server.displayName": "🇩🇪 Github Server",
    "server.description": "🇩🇪 The server to connect to. Only has to be set if
Github Enterprise is used.",
    "user.placeholder": "🇩🇪 Hans",
    "accessToken.placeholder": "🇩🇪 123"
}
```

Only existing parameters are translatable. If a credential parameter does not have a description in the English original, adding a translation for that non-existing parameter will not result in the translation being displayed - the parameter will need to be added in the English original first.

Translating the node translation file

> **Note**: All keys are optional. Missing translations trigger a fallback to the `en` locale strings.

Each node translation file is an object that allows for two keys, `header` and `nodeView`, which are the *sections* of each node translation.

The `header` section points to an object that may contain only two keys, `displayName` and `description`, matching the node's `description.displayName` and `description.description`.

```
export class Github implements INodeType {
    description: INodeTypeDescription = {
        displayName: 'GitHub', // key to use in translation
        description: 'Consume GitHub API', // key to use in translation
        name: 'github',
        icon: 'file:github.svg',
        group: ['input'],
        version: 1,
```

```
{
    "header": {
        "displayName": "🇩🇪 GitHub",
        "description": "🇩🇪 Consume GitHub API"
    }
}
```

Header text is used wherever the node's display name and description are needed:

In turn, the `nodeView` section points to an object containing translation keys that match the node's operational parameters, found in the `*.node.ts` and also found in `*Description.ts` files in the same dir.

```
export class Github implements INodeType {
    description: INodeTypeDescription = {
        displayName: 'GitHub',
        name: 'github',
        properties: [
            {
                displayName: 'Resource',
                name: 'resource', // key to use in translation
                type: 'options',
                options: [],
                default: 'issue',
                description: 'The resource to operate on.',
            },
```

```
{
    "nodeView.resource.displayName": "🇩🇪 Resource"
}
```

A node parameter allows for different translation keys depending on parameter type.

`string`, `number` and `boolean` parameters

Allowed keys: `displayName`, `description`, `placeholder`

```
{
    displayName: 'Repository Owner',
    name: 'owner', // key to use in translation
    type: 'string',
    required: true,
    placeholder: 'n8n-io',
    description: 'Owner of the repository.',
},
```

```
{
    "nodeView.owner.displayName": "🇩🇪 Repository Owner",
    "nodeView.owner.placeholder": "🇩🇪 n8n-io",
    "nodeView.owner.description": "🇩🇪 Owner of the repository"
}
```

`options` parameter

Allowed keys: `displayName`, `description`, `placeholder`

Allowed subkeys: `options.{optionName}.displayName` and `options.{optionName}.description`.

```
{
    displayName: 'Resource',
    name: 'resource',
    type: 'options',
    options: [
        {
            name: 'File',
            value: 'file', // key to use in translation
        },
        {
            name: 'Issue',
            value: 'issue', // key to use in translation
        },
    ],
    default: 'issue',
    description: 'Resource to operate on',
},
```

```
{
    "nodeView.resource.displayName": "🇩🇪 Resource",
    "nodeView.resource.description": "🇩🇪 Resource to operate on",
    "nodeView.resource.options.file.name": "🇩🇪 File",
    "nodeView.resource.options.issue.name": "🇩🇪 Issue"
}
```

For nodes whose credentials may be used in the HTTP Request node, an additional option `Custom API Call` is injected into the `Resource` and `Operation` parameters. Use the `__CUSTOM_API_CALL__` key to translate this additional option.

```
{
    "nodeView.resource.options.file.name": "🇩🇪 File",
    "nodeView.resource.options.issue.name": "🇩🇪 Issue",
    "nodeView.resource.options.__CUSTOM_API_CALL__.name": "🇩🇪 Custom API
Call"
}
```

`collection` and `fixedCollection` parameters

Allowed keys: `displayName`, `description`, `placeholder`, `multipleValueButtonText`

Example of `collection` parameter:

```
{
    displayName: 'Labels',
    name: 'labels', // key to use in translation
    type: 'collection',
    typeOptions: {
        multipleValues: true,
        multipleValueButtonText: 'Add Label',
    },
    displayOptions: {
        show: {
            operation: [
                'create',
            ],
            resource: [
                'issue',
            ],
        },
    },
    default: { 'label': '' },
    options: [
        {
            displayName: 'Label',
            name: 'label', // key to use in translation
            type: 'string',
            default: '',
            description: 'Label to add to issue',
        },
    ],
},
```

```
{
    "nodeView.labels.displayName": "🇩🇪 Labels",
    "nodeView.labels.multipleValueButtonText": "🇩🇪 Add Label",
    "nodeView.labels.options.label.displayName": "🇩🇪 Label",
    "nodeView.labels.options.label.description": "🇩🇪 Label to add to issue",
    "nodeView.labels.options.label.placeholder": "🇩🇪 Some placeholder"
}
```

Example of `fixedCollection` parameter:

```
{
    displayName: 'Additional Parameters',
    name: 'additionalParameters',
    placeholder: 'Add Parameter',
    description: 'Additional fields to add.',
    type: 'fixedCollection',
    default: {},
    displayOptions: {
        show: {
            operation: [
                'create',
                'delete',
                'edit',
            ],
            resource: [
                'file',
            ],
        },
    },
    options: [
        {
            name: 'author',
            displayName: 'Author',
            values: [
                {
                    displayName: 'Name',
                    name: 'name',
                    type: 'string',
                    default: '',
                    description: 'Name of the author of the commit',
                    placeholder: 'John',
                },
                {
                    displayName: 'Email',
                    name: 'email',
                    type: 'string',
                    default: '',
                    description: 'Email of the author of the commit',
                    placeholder: 'john@email.com',
                },
            ],
        },
    ],
}
```

```
{
    "nodeView.additionalParameters.displayName": "🇩🇪 Additional Parameters",
    "nodeView.additionalParameters.placeholder": "🇩🇪 Add Field",
    "nodeView.additionalParameters.options.author.displayName": "🇩🇪 Author",
    "nodeView.additionalParameters.options.author.values.name.displayName":
"🇩🇪 Name",
    "nodeView.additionalParameters.options.author.values.name.description":
"🇩🇪 Name of the author of the commit",
    "nodeView.additionalParameters.options.author.values.name.placeholder":
"🇩🇪 Jan",
    "nodeView.additionalParameters.options.author.values.email.displayName":
"🇩🇪 Email",
    "nodeView.additionalParameters.options.author.values.email.description":
"🇩🇪 Email of the author of the commit",
    "nodeView.additionalParameters.options.author.values.email.placeholder":
"🇩🇪 jan@n8n.io"
}
```

For information on **reusable dynamic text**, refer to the Addendum.

Building translations

Base text

When translating a base text file at `/packages/frontend/editor-ui/src/plugins/i18n/locales/{localeIdentifier}.json`:

1. Open a terminal:

```
export N8N_DEFAULT_LOCALE=de
pnpm start
```

2. Open another terminal:

```
export N8N_DEFAULT_LOCALE=de
cd packages/frontend/editor-ui
pnpm dev
```

Changing the base text file will trigger a rebuild of the client at `http://localhost:8080`.

Dynamic text

When translating a dynamic text file at `/packages/nodes-base/nodes/{node}/translations/{localeIdentifier}/{node}.json`,

1. Open a terminal:

```
export N8N_DEFAULT_LOCALE=de
pnpm start
```

2. Open another terminal:

```
export N8N_DEFAULT_LOCALE=de
cd packages/nodes-base
pnpm n8n-generate-translations
pnpm watch
```

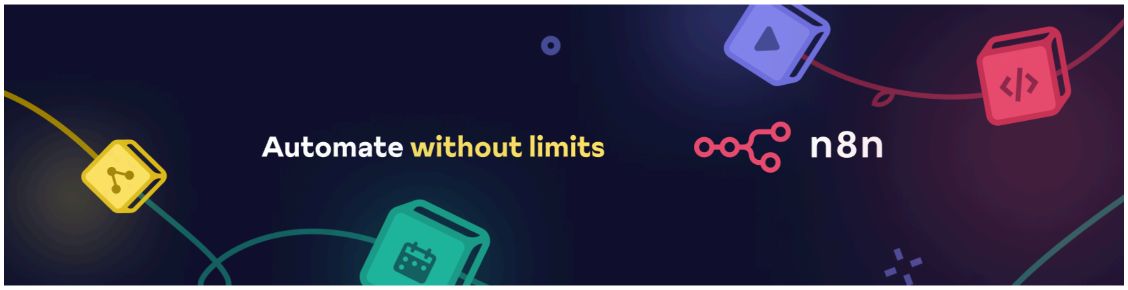
After changing the dynamic text file:

1. Stop and restart the first terminal.
2. Refresh the browser at `http://localhost:5678`

If a `headerText` section was changed, re-run `pnpm n8n-generate-translations` in `/nodes-base`.

> **Note**: To translate base and dynamic text simultaneously, run three terminals following the steps from both sections (first terminal running only once) and browse `http://localhost:8080`.

packages/node-dev/README.md



n8n.io - Workflow Automation

n8n-node-dev

Currently very simple and not very sophisticated CLI which makes it easier to create credentials and nodes in TypeScript for n8n.

```
npm install n8n-node-dev -g
```

Contents

Usage

The commandline tool can be started with `n8n-node-dev <COMMAND>`

Commands

The following commands exist:

build

Builds credentials and nodes in the current folder and copies them into the n8n custom extension folder (`~/.n8n/custom/`) unless destination path is overwritten with `--destination <FOLDER_PATH>`

When "–watch" gets set it starts in watch mode and automatically builds and copies files whenever they change. To stop press "ctrl + c".

new

Creates new basic credentials or node of the selected type to have a first starting point.

Create a node

The easiest way to create a new node is via the “n8n-node-dev” cli. It sets up all the basics.

A n8n node is a JavaScript file (normally written in TypeScript) which describes some basic information (like name, description, …) and also at least one method. Depending on which method gets implemented defines if it is a regular-, trigger- or webhook-node.

A simple regular node which:

- defines one node property
- sets its value to all items it receives

would look like this:

File named: `MyNode.node.ts`

```
import {
    IExecuteFunctions,
    INodeExecutionData,
    INodeType,
    INodeTypeDescription,
} from 'n8n-workflow';


export class MyNode implements INodeType {
    description: INodeTypeDescription = {
        displayName: 'My Node',
        name: 'myNode',
        group: ['transform'],
        version: 1,
        description: 'Adds "myString" on all items to defined value.',
        defaults: {
            name: 'My Node',
            color: '#772244',
        },
        inputs: ['main'],
        outputs: ['main'],
        properties: [
            // Node properties which the user gets displayed and
            // can change on the node.
            {
                displayName: 'My String',
                name: 'myString',
                type: 'string',
                default: '',
                placeholder: 'Placeholder value',
                description: 'The description text',
            }
        ]
    };


    async execute(this: IExecuteFunctions): Promise<INodeExecutionData[][]> {

        const items = this.getInputData();

        let item: INodeExecutionData;
        let myString: string;

        // Itterates over all input items and add the key "myString" with the
        // value the parameter "myString" resolves to.
        // (This could be a different value for each item in case it contains
an expression)
        for (let itemIndex = 0; itemIndex < items.length; itemIndex++) {
            myString = this.getNodeParameter('myString', itemIndex, '') as
string;
            item = items[itemIndex];

            item.json['myString'] = myString;
        }

        return [items];

    }
}
```

The “description” property has to be set on all nodes because it contains all the base information. Additionally all nodes have to have exactly one of the following methods defined which contains the actual logic:

Regular node

Method is called when the workflow gets executed

- `execute`: Executed once no matter how many items

By default, `execute` should always be used, especially when creating a third-party integration. The reason for this is that it provides much more flexibility and allows, for example, returning a different number of items than it received as input. This becomes crucial when a node needs to query data such as *return all users*. In such cases, the node typically receives only one input item but returns as many items as there are users. Therefore, when in doubt, it is recommended to use `execute`!

Trigger node

Method is called once when the workflow gets activated. It can then trigger workflow runs and provide the necessary data by itself.

- `trigger`

Webhook node

Method is called when webhook gets called.

- `webhook`

Node Type

Property overview

- **description** [required]: Describes the node like its name, properties, hooks, … see `Node Type Description` bellow.
- **execute** [optional]: Method is called when the workflow gets executed (once).
- **hooks** [optional]: The hook methods.
- **methods** [optional]: Additional methods. Currently only “loadOptions” exists which allows loading options for parameters from external services
- **trigger** [optional]: Method is called once when the workflow gets activated.
- **webhook** [optional]: Method is called when webhook gets called.
- **webhookMethods** [optional]: Methods to setup webhooks on external services.

Node Type Description

The following properties can be set in the node description:

- **credentials** [optional]: Credentials the node requests access to
- **defaults** [required]: Default “name” and “color” to set on node when it gets created
- **displayName** [required]: Name to display users in Editor UI
- **description** [required]: Description to display users in Editor UI
- **group** [required]: Node group for example “transform” or “trigger”
- **hooks** [optional]: Methods to execute at different points in time like when the workflow gets activated or deactivated
- **icon** [optional]: Icon to display (can be an icon or a font awesome icon)
- **inputs** [required]: Types of inputs the node has (currently only “main” exists) and the amount
- **outputs** [required]: Types of outputs the node has (currently only “main” exists) and the amount
- **outputNames** [optional]: In case a node has multiple outputs, names can be set that users know what data to expect

- **maxNodes** [optional]: If an unlimited number of nodes of that type cannot exist in a workflow, the max-amount can be specified
- **name** [required]: Name of the node (for n8n to use internally, in camelCase)
- **properties** [required]: Properties which get displayed in the Editor UI and can be set by the user
- **subtitle** [optional]: Text which should be displayed underneath the name of the node in the Editor UI (can be an expression)
- **version** [required]: Version of the node. Currently always “1” (integer). For future usage, does not get used yet
- **webhooks** [optional]: Webhooks the node should listen to

Node Properties

The following properties can be set in the node properties:

- **default** [required]: Default value of the property
- **description** [required]: Description that is displayed to users in the Editor UI
- **displayName** [required]: Name that is displayed to users in the Editor UI
- **displayOptions** [optional]: Defines logic to decide if a property should be displayed or not
- **name** [required]: Name of the property (for n8n to use internally, in camelCase)
- **options** [optional]: The options the user can select when type of property is “collection”, “fixedCollection” or “options”
- **placeholder** [optional]: Placeholder text that is displayed to users in the Editor UI
- **type** [required]: Type of the property. If it is for example a “string”, “number”, …
- **typeOptions** [optional]: Additional options for type. Like for example the min or max value of a number
- **required** [optional]: Defines if the value has to be set or if it can stay empty

Node Property Options

The following properties can be set in the node property options:

All properties are optional. However, most only work when the node-property is of a specfic type.

- **alwaysOpenEditWindow** [type: json]: If set then the “Editor Window” will always open when the user tries to edit the field. Helpful if long text is typically used in the property
- **loadOptionsMethod** [type: options]: Method to use to load options from an external service
- **maxValue** [type: number]: Maximum value of the number
- **minValue** [type: number]: Minimum value of the number
- **multipleValues** [type: all]: If set the property gets turned into an Array and the user can add multiple values

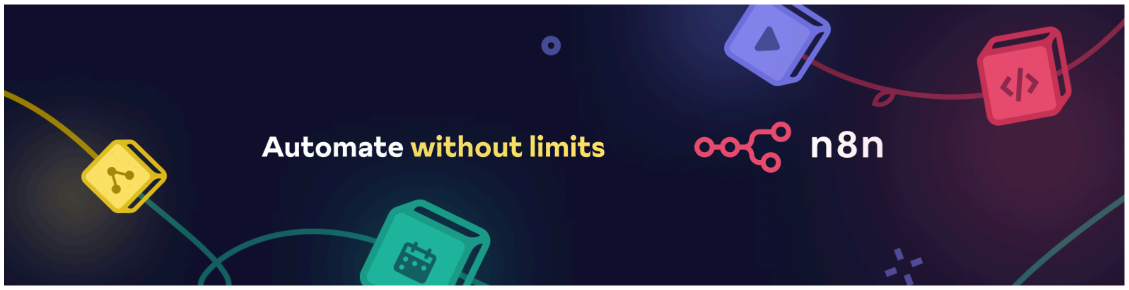
- **multipleValueButtonText** [type: all]: Custom text for add button in case "multipleValues" were set
- **numberPrecision** [type: number]: The precision of the number. By default, it is "0" and will only allow integers
- **password** [type: string]: If a password field should be displayed (normally only used by credentials because all node data is not encrypted and gets saved in clear-text)
- **rows** [type: string]: Number of rows the input field should have. By default it is "1"

License

You can find the license information here

---

packages/nodes-base/README.md



n8n.io - Workflow Automation

n8n-nodes-base

The nodes which are included by default in n8n

```
npm install n8n-nodes-base -g
```

License

You can find the license information here

---

packages/nodes-base/nodes/Stripe/README.md

All Stripe webhook events are taken from docs: https://stripe.com/docs/api/events/types#event_types

To get the entire list of events as a JS array, scrape the website:

1. manually add the id #event-types to `<ul>` that contains all event types

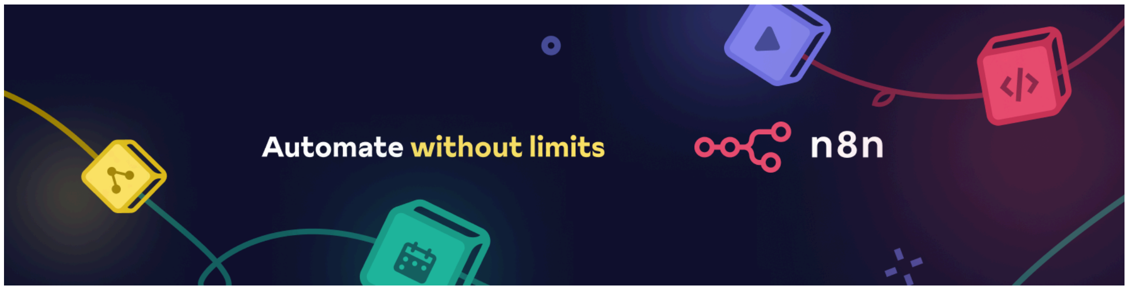
2. copy-paste the function in the JS console
3. the result is copied into in the clipboard
4. paste the prepared array in StripeTrigger.node.ts

```
types = [];
$$('ul#event-types li').forEach((el) => {
    const value = el.querySelector('.method-list-item-label-name').innerText;

    types.push({
        name: value
            .replace(/(\.|_)/, ' ')
            .split(' ')
            .map((s) => s.charAt(0).toUpperCase() + s.substring(1))
            .join(' '),
        value,
        description:
el.querySelector('.method-list-item-description').innerText,
    });
});
copy(types);
```

---

packages/workflow/README.md



n8n.io - Workflow Automation

n8n-workflow

Workflow base code for n8n

```
npm install n8n-workflow
```

License

You can find the license information here

---

test-workflows/README.md

test-workflows

n8n workflows used for testing nodes
