NodeSpark + NodeSparkHub
Combined User Manual
Version B (20 Tutorials + Full Reference + Dictionaries)

Generated: January 10, 2026

Overview

This manual is intentionally comprehensive for a Store-ready release. It includes: a beginner-friendly introduction, end-to-end iOS and macOS instructions, a complete mode reference, 20 tutorial workflows, and dictionaries that explain parameter keys and screens.

If you do not know JSON:

```
Start with Section 1 and Tutorial 1. You can be productive without becoming a developer.
```

1. Concepts

1.1 What a workflow is

A workflow is a chain of nodes. Each node performs one small job and passes a payload to the next node.

1.2 JSON crash course

JSON is how NodeSpark represents data. Learn these rules and you can use every feature.

- Objects: { } — key/value pairs.
- Arrays: [ ] — ordered lists.
- Strings: "text" — always double quotes.
- Numbers: 1, 2.5 — no quotes.
- Booleans: true / false.

Copy/paste JSON example

```
{
  "input": {
    "text": "Hello",
    "count": 2,
    "ok": true,
    "tags": ["a", "b"]
  }
}
```

1.3 How to read Run History

1. Open Run History for the most recent run.
2. Start at the first node and look at its Output.
3. Compare that output to the next node's Input.
4. The first mismatch is where you fix the workflow.

2. NodeSpark (iOS) - Screen-by-screen

2.1 Core screens

These are the screens you will use most often:

- Dashboard: manage workflows and templates.
- Workflow Editor: canvas for building flows.
- Node Inspector: configure modes and parameters.
- Run History: inspect inputs/outputs and errors.
- Settings: preferences and connections.

2.2 Typical daily workflow

5. Create or open a workflow.
6. Quick Run it with a small input.
7. Inspect Run History.
8. Adjust parameters until output shape is stable.
9. Save as template if reusable.

3. NodeSparkHub (macOS) - Screen-by-screen

3.1 Hub server controls

10. Start server.
11. Confirm LAN URL.
12. Enable pairing lock if desired.
13. Set webhook secret.
14. Optionally enable TLS.

3.2 Webhook testing

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '{ "input": { "text": "hello" } }'
```

4. Using both together

4.1 Recommended deployment model

- Develop on iOS using Quick Run.
- Deploy to Hub for schedules/webhooks and long-running automation.
- Debug on Hub when integrations fail.

4.2 Hub endpoints (inventory)

- /api/v1
- /api/v1/
- /clients
- /clients/ping
- /devices
- /devices/
- /devices/checkin
- /health
- /pair
- /result
- /run
- /runs
- /runs/
- /runs/latest
- /schedules
- /schedules/
- /status
- /tls/fingerprint
- /trace
- /trigger/
- /workers
- /workers/
- /workflows
- /workflows/

5. Full Reference — Modes

This section covers every mode string detected in your builds. Even if you do not use a mode today, skimming this section will make the editor feel predictable.

append_log — append_log

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

15. Open the node Inspector.
16. Set Mode to this value.
17. Fill required fields (the inspector will usually indicate required inputs).
18. Run Quick Run with a small input payload.
19. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

appevent — appevent

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

20. Open the node Inspector.
21. Set Mode to this value.
22. Fill required fields (the inspector will usually indicate required inputs).

23. Run Quick Run with a small input payload.
24. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

bool — bool

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

25. Open the node Inspector.
26. Set Mode to this value.
27. Fill required fields (the inspector will usually indicate required inputs).
28. Run Quick Run with a small input payload.
29. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

calendarevent — calendarevent

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

30. Open the node Inspector.
31. Set Mode to this value.
32. Fill required fields (the inspector will usually indicate required inputs).
33. Run Quick Run with a small input payload.
34. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

clipboard — clipboard

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

35. Open the node Inspector.
36. Set Mode to this value.
37. Fill required fields (the inspector will usually indicate required inputs).
38. Run Quick Run with a small input payload.
39. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

condition — condition

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

40. Open the node Inspector.
41. Set Mode to this value.
42. Fill required fields (the inspector will usually indicate required inputs).
43. Run Quick Run with a small input payload.
44. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

contains — Contains

Returns true if a string contains a substring (used for gating / branching).

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

45. Open the node Inspector.
46. Set Mode to this value.
47. Fill required fields (the inspector will usually indicate required inputs).
48. Run Quick Run with a small input payload.
49. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

cron — cron

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

50. Open the node Inspector.
51. Set Mode to this value.
52. Fill required fields (the inspector will usually indicate required inputs).
53. Run Quick Run with a small input payload.
54. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }
```

```
Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

delay — Delay

Waits a specified duration before continuing (useful for rate limits).

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

55. Open the node Inspector.
56. Set Mode to this value.
57. Fill required fields (the inspector will usually indicate required inputs).
58. Run Quick Run with a small input payload.
59. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

double — double

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

60. Open the node Inspector.
61. Set Mode to this value.
62. Fill required fields (the inspector will usually indicate required inputs).
63. Run Quick Run with a small input payload.
64. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

echo — Echo

Writes a value to output/log for debugging.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

65. Open the node Inspector.
66. Set Mode to this value.
67. Fill required fields (the inspector will usually indicate required inputs).
68. Run Quick Run with a small input payload.
69. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.

- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

ends_with — ends_with

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

70. Open the node Inspector.
71. Set Mode to this value.
72. Fill required fields (the inspector will usually indicate required inputs).
73. Run Quick Run with a small input payload.
74. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

equals — Equals

Compares two values for equality (often strings/numbers).

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

75. Open the node Inspector.
76. Set Mode to this value.

77. Fill required fields (the inspector will usually indicate required inputs).
78. Run Quick Run with a small input payload.
79. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

extract_tasks — extract_tasks

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

80. Open the node Inspector.
81. Set Mode to this value.
82. Fill required fields (the inspector will usually indicate required inputs).
83. Run Quick Run with a small input payload.
84. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

fri — fri

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

85. Open the node Inspector.
86. Set Mode to this value.
87. Fill required fields (the inspector will usually indicate required inputs).
88. Run Quick Run with a small input payload.
89. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

gt — Greater Than

Numeric comparison for branching.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

90. Open the node Inspector.
91. Set Mode to this value.
92. Fill required fields (the inspector will usually indicate required inputs).
93. Run Quick Run with a small input payload.
94. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

http — http

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

95. Open the node Inspector.
96. Set Mode to this value.
97. Fill required fields (the inspector will usually indicate required inputs).
98. Run Quick Run with a small input payload.
99. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

http_get — HTTP GET

Fetch JSON (or text) from a URL. Common for status endpoints and simple APIs.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

100. Open the node Inspector.
101. Set Mode to this value.
102. Fill required fields (the inspector will usually indicate required inputs).
103. Run Quick Run with a small input payload.
104. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "url": "https://example.com/status" } }

Example output:
{ "http": { "status": 200, "body": { "...": "..." } } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

if — if

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

105. Open the node Inspector.
106. Set Mode to this value.
107. Fill required fields (the inspector will usually indicate required inputs).
108. Run Quick Run with a small input payload.
109. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }
```

```
Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

int — int

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

110. Open the node Inspector.
111. Set Mode to this value.
112. Fill required fields (the inspector will usually indicate required inputs).
113. Run Quick Run with a small input payload.
114. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

is_empty — is_empty

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

115. Open the node Inspector.
116. Set Mode to this value.
117. Fill required fields (the inspector will usually indicate required inputs).
118. Run Quick Run with a small input payload.
119. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

is_not_empty — is_not_empty

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

120. Open the node Inspector.
121. Set Mode to this value.
122. Fill required fields (the inspector will usually indicate required inputs).
123. Run Quick Run with a small input payload.
124. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.

- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

lowercase — Lowercase

Converts a string to lower case.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

125. Open the node Inspector.
126. Set Mode to this value.
127. Fill required fields (the inspector will usually indicate required inputs).
128. Run Quick Run with a small input payload.
129. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

lt — Less Than

Numeric comparison for branching.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

130. Open the node Inspector.
131. Set Mode to this value.

132. Fill required fields (the inspector will usually indicate required inputs).
133. Run Quick Run with a small input payload.
134. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

mail — mail

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

135. Open the node Inspector.
136. Set Mode to this value.
137. Fill required fields (the inspector will usually indicate required inputs).
138. Run Quick Run with a small input payload.
139. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

manual — manual

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

140. Open the node Inspector.
141. Set Mode to this value.
142. Fill required fields (the inspector will usually indicate required inputs).
143. Run Quick Run with a small input payload.
144. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

map_fields — Map Fields

Map/rename fields from one object shape to another (hub canonical mode).

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

145. Open the node Inspector.
146. Set Mode to this value.
147. Fill required fields (the inspector will usually indicate required inputs).
148. Run Quick Run with a small input payload.
149. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

mapfields — mapfields

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

150. Open the node Inspector.
151. Set Mode to this value.
152. Fill required fields (the inspector will usually indicate required inputs).
153. Run Quick Run with a small input payload.
154. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

merge_json — Merge JSON

Merge an object into the current payload (hub canonical mode).

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

155. Open the node Inspector.
156. Set Mode to this value.
157. Fill required fields (the inspector will usually indicate required inputs).
158. Run Quick Run with a small input payload.
159. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

mergejson — mergejson

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

160. Open the node Inspector.
161. Set Mode to this value.
162. Fill required fields (the inspector will usually indicate required inputs).
163. Run Quick Run with a small input payload.
164. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }
```

```
Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

messages — messages

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

165. Open the node Inspector.
166. Set Mode to this value.
167. Fill required fields (the inspector will usually indicate required inputs).
168. Run Quick Run with a small input payload.
169. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

mon — mon

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

170. Open the node Inspector.
171. Set Mode to this value.
172. Fill required fields (the inspector will usually indicate required inputs).
173. Run Quick Run with a small input payload.
174. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

monthly — monthly

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

175. Open the node Inspector.
176. Set Mode to this value.
177. Fill required fields (the inspector will usually indicate required inputs).
178. Run Quick Run with a small input payload.
179. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.

- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

not_contains — not_contains

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

180. Open the node Inspector.
181. Set Mode to this value.
182. Fill required fields (the inspector will usually indicate required inputs).
183. Run Quick Run with a small input payload.
184. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

not_equals — not_equals

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

185. Open the node Inspector.
186. Set Mode to this value.

187. Fill required fields (the inspector will usually indicate required inputs).
188. Run Quick Run with a small input payload.
189. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

notification — Notification

Shows a local notification with a title/body built from payload values.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

190. Open the node Inspector.
191. Set Mode to this value.
192. Fill required fields (the inspector will usually indicate required inputs).
193. Run Quick Run with a small input payload.
194. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

openurl — openurl

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

195. Open the node Inspector.
196. Set Mode to this value.
197. Fill required fields (the inspector will usually indicate required inputs).
198. Run Quick Run with a small input payload.
199. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

read_variable — Read Variable

Reads a previously set variable and inserts it into the payload.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

200. Open the node Inspector.
201. Set Mode to this value.
202. Fill required fields (the inspector will usually indicate required inputs).
203. Run Quick Run with a small input payload.
204. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

reminder — reminder

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

205. Open the node Inspector.
206. Set Mode to this value.
207. Fill required fields (the inspector will usually indicate required inputs).
208. Run Quick Run with a small input payload.
209. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

rewrite_friendly — rewrite_friendly

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

210. Open the node Inspector.
211. Set Mode to this value.
212. Fill required fields (the inspector will usually indicate required inputs).
213. Run Quick Run with a small input payload.
214. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

run_subflow — run_subflow

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

215. Open the node Inspector.
216. Set Mode to this value.
217. Fill required fields (the inspector will usually indicate required inputs).
218. Run Quick Run with a small input payload.
219. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }
```

```
Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

runshortcut — runshortcut

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

220. Open the node Inspector.
221. Set Mode to this value.
222. Fill required fields (the inspector will usually indicate required inputs).
223. Run Quick Run with a small input payload.
224. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

runsubflow — runsubflow

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

225. Open the node Inspector.
226. Set Mode to this value.
227. Fill required fields (the inspector will usually indicate required inputs).
228. Run Quick Run with a small input payload.
229. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

sat — sat

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

230. Open the node Inspector.
231. Set Mode to this value.
232. Fill required fields (the inspector will usually indicate required inputs).
233. Run Quick Run with a small input payload.
234. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.

- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

schedule — schedule

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

235. Open the node Inspector.
236. Set Mode to this value.
237. Fill required fields (the inspector will usually indicate required inputs).
238. Run Quick Run with a small input payload.
239. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

share — share

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

240. Open the node Inspector.
241. Set Mode to this value.

242. Fill required fields (the inspector will usually indicate required inputs).
243. Run Quick Run with a small input payload.
244. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

starts_with — starts_with

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

245. Open the node Inspector.
246. Set Mode to this value.
247. Fill required fields (the inspector will usually indicate required inputs).
248. Run Quick Run with a small input payload.
249. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

string — string

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

250. Open the node Inspector.
251. Set Mode to this value.
252. Fill required fields (the inspector will usually indicate required inputs).
253. Run Quick Run with a small input payload.
254. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

summarize — summarize

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

255. Open the node Inspector.
256. Set Mode to this value.
257. Fill required fields (the inspector will usually indicate required inputs).
258. Run Quick Run with a small input payload.
259. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

sun — sun

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

260. Open the node Inspector.
261. Set Mode to this value.
262. Fill required fields (the inspector will usually indicate required inputs).
263. Run Quick Run with a small input payload.
264. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

thu — thu

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

265. Open the node Inspector.
266. Set Mode to this value.
267. Fill required fields (the inspector will usually indicate required inputs).
268. Run Quick Run with a small input payload.
269. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

transform_json — Transform JSON (Template)

Build a new JSON object from the current payload using a template; supports variable substitution.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

270. Open the node Inspector.
271. Set Mode to this value.
272. Fill required fields (the inspector will usually indicate required inputs).
273. Run Quick Run with a small input payload.
274. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "first": "Jane", "last": "Doe" } }
```

```
Example transform template:
{
  "person": {
    "fullName": "{{input.first}} {{input.last}}"
  }
}

Example output:
{ "person": { "fullName": "Jane Doe" } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

trim — Trim

Removes leading/trailing whitespace from a string.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

275. Open the node Inspector.
276. Set Mode to this value.
277. Fill required fields (the inspector will usually indicate required inputs).
278. Run Quick Run with a small input payload.
279. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

trim_whitespace — trim_whitespace

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

280. Open the node Inspector.
281. Set Mode to this value.
282. Fill required fields (the inspector will usually indicate required inputs).
283. Run Quick Run with a small input payload.
284. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

tue — tue

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

285. Open the node Inspector.
286. Set Mode to this value.
287. Fill required fields (the inspector will usually indicate required inputs).
288. Run Quick Run with a small input payload.
289. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

uppercase — Uppercase

Converts a string to upper case.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

290. Open the node Inspector.
291. Set Mode to this value.
292. Fill required fields (the inspector will usually indicate required inputs).
293. Run Quick Run with a small input payload.
294. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

webhook — Webhook Trigger

Starts a workflow when the hub receives an HTTP POST with JSON. Use a secret header for security.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

295. Open the node Inspector.
296. Set Mode to this value.
297. Fill required fields (the inspector will usually indicate required inputs).
298. Run Quick Run with a small input payload.
299. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

wed — wed

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

300. Open the node Inspector.
301. Set Mode to this value.
302. Fill required fields (the inspector will usually indicate required inputs).
303. Run Quick Run with a small input payload.
304. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }
```

```
Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

weekly — weekly

Mode description: refer to node inspector for the authoritative behavior in your build.

What this mode is good for

- Building predictable outputs for downstream steps.
- Reducing manual copy/paste work.
- Standardizing data shapes (especially before AI or HTTP calls).

How to configure it (beginner steps)

305. Open the node Inspector.
306. Set Mode to this value.
307. Fill required fields (the inspector will usually indicate required inputs).
308. Run Quick Run with a small input payload.
309. Verify output in Run History; iterate.

Input and output (examples)

```
Example input:
{ "input": { "text": "Hello" } }

Example output:
{ "output": { "result": "..." } }
```

Troubleshooting for this mode

- If output is empty: verify you referenced existing keys and that they contain values.
- If the run stops: look at the node error; it often indicates a missing required parameter or type mismatch.
- If behavior differs between iOS and Hub: check the mode canonicalization and the hub engine support list.

6. Tutorial Workflows (20)

These tutorials are designed to be followed exactly. They are intentionally verbose to prevent beginner errors.

Tutorial 1: Daily Briefing Summary

Summarize long text into a structured briefing object and notify yourself.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: summarize
- Logic: transform_json
- Action: notification

Build steps (detailed)

310. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
311. Add nodes exactly in the order shown. Connect them left-to-right.
312. For each node: open Inspector → set Mode → fill parameters → close Inspector.
313. Use Quick Run with the sample JSON. Keep your first run small and simple.
314. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "input": {
    "text": "Paste a long article or notes here."
  }
}
```

Expected output shape

```
{
  "briefing": {
    "title": "…",
    "bullets": ["…","…","…"],
    "nextActions": ["…"]
  }
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

315. Start Hub server and confirm LAN URL.
316. Pair iOS to Hub if pairing is enabled.
317. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 2: Meeting Notes to Action Items

Convert raw meeting notes into a JSON task list and copy it to the clipboard.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: extract tasks as JSON
- Logic: validate_required_fields
- Action: copy_to_clipboard

Build steps (detailed)

318. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
319. Add nodes exactly in the order shown. Connect them left-to-right.
320. For each node: open Inspector → set Mode → fill parameters → close Inspector.
321. Use Quick Run with the sample JSON. Keep your first run small and simple.
322. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "input": {
    "notes": "Attendees: ... Decisions: ... Action: John to ... by Friday
..."
  }
}
```

Expected output shape

```
{
  "tasks": [
    { "title": "…", "owner": "…", "due": "YYYY-MM-DD", "priority":
"low|med|high" }
  ]
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

323. Start Hub server and confirm LAN URL.
324. Pair iOS to Hub if pairing is enabled.
325. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 3: Email Draft Generator

Generate a professional email (subject + body) from bullet points using a Prompt Profile.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: draft email
- Logic: transform_json
- Action: copy_to_clipboard

Build steps (detailed)

326. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
327. Add nodes exactly in the order shown. Connect them left-to-right.
328. For each node: open Inspector → set Mode → fill parameters → close Inspector.
329. Use Quick Run with the sample JSON. Keep your first run small and simple.
330. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "input": {
    "to": "Vendor Name",
    "purpose": "Request an updated quote",
    "bullets": ["Need pricing for ...", "Timeline is ...", "Please include
..."]
  }
}
```

Expected output shape

```
{
  "email": {
    "subject": "…",
    "body": "Dear …\n\n…"
  }
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
```

```
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

331. Start Hub server and confirm LAN URL.
332. Pair iOS to Hub if pairing is enabled.
333. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 4: Inbox Cleaner (Classification)

Classify messages into triage categories.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: classify
- Logic: group_by
- Action: copy_to_clipboard

Build steps (detailed)

334. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
335. Add nodes exactly in the order shown. Connect them left-to-right.
336. For each node: open Inspector → set Mode → fill parameters → close Inspector.
337. Use Quick Run with the sample JSON. Keep your first run small and simple.
338. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "input": {
    "messages": [
      {"from":"…","subject":"…","snippet":"…"},
      {"from":"…","subject":"…","snippet":"…"}
    ]
  }
}
```

Expected output shape

```
{
  "triage": {
    "urgent": [],
    "respond": [],
    "readLater": [],
    "archive": []
  }
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

339. Start Hub server and confirm LAN URL.
340. Pair iOS to Hub if pairing is enabled.
341. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 5: Website Watch via HTTP GET

Fetch a URL daily and summarize what changed.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: schedule_daily (Hub recommended)
- HTTP: http_get
- AI: summarize
- Action: notification

Build steps (detailed)

342. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
343. Add nodes exactly in the order shown. Connect them left-to-right.
344. For each node: open Inspector → set Mode → fill parameters → close Inspector.
345. Use Quick Run with the sample JSON. Keep your first run small and simple.
346. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "url": "https://example.com/status" } }
```

Expected output shape

```
{ "summary": "…", "important": true }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

347. Start Hub server and confirm LAN URL.
348. Pair iOS to Hub if pairing is enabled.
349. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 6: Webhook Intake - Simple Form

Trigger from external JSON and produce a confirmation response.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: webhook (Hub)
- Logic: validate_required_fields
- AI: rewrite
- Action: notification

Build steps (detailed)

350. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
351. Add nodes exactly in the order shown. Connect them left-to-right.
352. For each node: open Inspector → set Mode → fill parameters → close Inspector.
353. Use Quick Run with the sample JSON. Keep your first run small and simple.
354. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "form": {
    "name": "Alice",
    "email": "alice@example.com",
    "message": "I need help with…"
  }
}
```

Expected output shape

```
{ "ok": true, "ticketTitle": "…", "replyDraft": "…" }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

355. Start Hub server and confirm LAN URL.

356. Pair iOS to Hub if pairing is enabled.

357. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 7: JSON Normalizer (Transform Template)

Reshape input into a stable schema using transform_json.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Logic: transform_json

Build steps (detailed)

358. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
359. Add nodes exactly in the order shown. Connect them left-to-right.
360. For each node: open Inspector → set Mode → fill parameters → close Inspector.
361. Use Quick Run with the sample JSON. Keep your first run small and simple.
362. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{
  "input": {
    "user": {"first":"Jane","last":"Doe"},
    "contact": {"email":"jane@example.com"},
    "tags": ["trial","ios"]
  }
}
```

Expected output shape

```
{
  "person": {
    "fullName": "Jane Doe",
    "email": "jane@example.com",
    "tags": ["trial","ios"]
  }
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

363. Start Hub server and confirm LAN URL.
364. Pair iOS to Hub if pairing is enabled.
365. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 8: Condition Gate (If/Else)

Branch workflow based on keywords or priority.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Logic: contains/equals
- AI: summarize
- Action: notification

Build steps (detailed)

366. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
367. Add nodes exactly in the order shown. Connect them left-to-right.
368. For each node: open Inspector → set Mode → fill parameters → close Inspector.
369. Use Quick Run with the sample JSON. Keep your first run small and simple.
370. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "URGENT: ..." } }
```

Expected output shape

```
{ "route": "urgent", "summary": "…" }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

371. Start Hub server and confirm LAN URL.
372. Pair iOS to Hub if pairing is enabled.
373. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 9: Reminder Builder

Parse natural language into a reminder and create a device action (if supported).

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: parse reminder JSON
- Action: notification/reminder

Build steps (detailed)

374. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
375. Add nodes exactly in the order shown. Connect them left-to-right.
376. For each node: open Inspector → set Mode → fill parameters → close Inspector.
377. Use Quick Run with the sample JSON. Keep your first run small and simple.
378. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "Remind me to call the vendor tomorrow at 3pm." } }
```

Expected output shape

```
{ "deviceActions": [ { "type": "reminder", "title": "Call vendor", "due":
"…" } ] }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

379. Start Hub server and confirm LAN URL.
380. Pair iOS to Hub if pairing is enabled.
381. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 10: Calendar Event Builder

Extract event fields from text and create a calendar event (if supported).

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: extract event JSON
- Action: calendar_event

Build steps (detailed)

382. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
383. Add nodes exactly in the order shown. Connect them left-to-right.
384. For each node: open Inspector → set Mode → fill parameters → close Inspector.
385. Use Quick Run with the sample JSON. Keep your first run small and simple.
386. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "Schedule a 30 min sync with Bob next Tuesday at
10am." } }
```

Expected output shape

```
{ "event": { "title":"…", "start":"…", "end":"…", "location":"…" } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

387. Start Hub server and confirm LAN URL.
388. Pair iOS to Hub if pairing is enabled.
389. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 11: Clipboard Transformer

Clean text using basic string nodes and copy output to clipboard.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Logic: trim/lowercase/uppercase
- Action: copy_to_clipboard

Build steps (detailed)

390. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
391. Add nodes exactly in the order shown. Connect them left-to-right.
392. For each node: open Inspector → set Mode → fill parameters → close Inspector.
393. Use Quick Run with the sample JSON. Keep your first run small and simple.
394. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "   EXTRA   Spaces   " } }
```

Expected output shape

```
{ "output": { "text": "extra   spaces" } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

395. Start Hub server and confirm LAN URL.
396. Pair iOS to Hub if pairing is enabled.
397. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 12: URL Builder + Open

Construct a URL from inputs and open it.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Logic: transform_json
- Action: open_url

Build steps (detailed)

398. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
399. Add nodes exactly in the order shown. Connect them left-to-right.
400. For each node: open Inspector → set Mode → fill parameters → close Inspector.
401. Use Quick Run with the sample JSON. Keep your first run small and simple.
402. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "query": "workflow automation" } }
```

Expected output shape

```
{ "url": "https://www.google.com/search?q=workflow%20automation" }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

403. Start Hub server and confirm LAN URL.
404. Pair iOS to Hub if pairing is enabled.
405. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 13: Hub Remote Run from iOS

Run a workflow on Hub and inspect logs on both sides.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Hub: run
- Hub: run history

Build steps (detailed)

406. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
407. Add nodes exactly in the order shown. Connect them left-to-right.
408. For each node: open Inspector → set Mode → fill parameters → close Inspector.
409. Use Quick Run with the sample JSON. Keep your first run small and simple.
410. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "Run this on the hub." } }
```

Expected output shape

```
{ "ok": true, "output": { "…" : "…" } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

411. Start Hub server and confirm LAN URL.
412. Pair iOS to Hub if pairing is enabled.
413. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 14: Pairing Lock Walkthrough

Turn on pairing lock and verify unauthorized requests fail.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Hub: enable pairing lock
- Hub: generate pairing code
- iOS: pair
- Test: curl unauthenticated

Build steps (detailed)

414. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
415. Add nodes exactly in the order shown. Connect them left-to-right.
416. For each node: open Inspector → set Mode → fill parameters → close Inspector.
417. Use Quick Run with the sample JSON. Keep your first run small and simple.
418. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{}
```

Expected output shape

```
{}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

419. Start Hub server and confirm LAN URL.
420. Pair iOS to Hub if pairing is enabled.
421. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 15: Webhook Secret Rotation

Rotate webhook secrets without breaking callers.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Hub: rotate secret
- Update caller header
- Test webhook trigger

Build steps (detailed)

422. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
423. Add nodes exactly in the order shown. Connect them left-to-right.
424. For each node: open Inspector → set Mode → fill parameters → close Inspector.
425. Use Quick Run with the sample JSON. Keep your first run small and simple.
426. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{}
```

Expected output shape

```
{}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

427. Start Hub server and confirm LAN URL.
428. Pair iOS to Hub if pairing is enabled.
429. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 16: TLS Enablement (LAN HTTPS)

Enable HTTPS on Hub and connect iOS securely.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Hub: import .p12
- Hub: enable TLS
- Restart server
- iOS: update URL

Build steps (detailed)

430. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
431. Add nodes exactly in the order shown. Connect them left-to-right.
432. For each node: open Inspector → set Mode → fill parameters → close Inspector.
433. Use Quick Run with the sample JSON. Keep your first run small and simple.
434. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{}
```

Expected output shape

```
{}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

435. Start Hub server and confirm LAN URL.
436. Pair iOS to Hub if pairing is enabled.
437. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 17: AI Rewrite - Professional Tone

Rewrite informal text to a professional message.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: rewrite
- Action: copy_to_clipboard

Build steps (detailed)

438. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
439. Add nodes exactly in the order shown. Connect them left-to-right.
440. For each node: open Inspector → set Mode → fill parameters → close Inspector.
441. Use Quick Run with the sample JSON. Keep your first run small and simple.
442. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "hey can u send that asap thx" } }
```

Expected output shape

```
{ "output": { "text": "Hello, could you please send that at your earliest convenience? Thank you." } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

443. Start Hub server and confirm LAN URL.
444. Pair iOS to Hub if pairing is enabled.
445. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 18: Structured Extraction - Contact Card

Extract a contact card from unstructured text.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: strict JSON extraction
- Logic: validate_required_fields

Build steps (detailed)

446. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
447. Add nodes exactly in the order shown. Connect them left-to-right.
448. For each node: open Inspector → set Mode → fill parameters → close Inspector.
449. Use Quick Run with the sample JSON. Keep your first run small and simple.
450. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "Sarah Connor, sarah@example.com, (555) 123-4567" } }
```

Expected output shape

```
{ "contact": { "name":"Sarah Connor","email":"…","phone":"…" } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

451. Start Hub server and confirm LAN URL.
452. Pair iOS to Hub if pairing is enabled.
453. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 19: Multi-step Agent: Research + Summary

Create a structured report using an agent node (if enabled).

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- AI: agent
- Logic: transform_json
- Action: copy_to_clipboard

Build steps (detailed)

454. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
455. Add nodes exactly in the order shown. Connect them left-to-right.
456. For each node: open Inspector → set Mode → fill parameters → close Inspector.
457. Use Quick Run with the sample JSON. Keep your first run small and simple.
458. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "topic": "Predictive maintenance KPIs" } }
```

Expected output shape

```
{ "report": { "outline":[...], "summary":"...", "nextSteps":[...] } }
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

459. Start Hub server and confirm LAN URL.
460. Pair iOS to Hub if pairing is enabled.
461. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
```

```
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

Tutorial 20: Automation Hand-off: Device Actions

Output a deviceActions array that can be consumed by action handlers.

Prerequisites

- NodeSpark installed.
- Run History available.
- If using Hub: Hub running on same Wi-Fi and you know the LAN URL.

Node list (build order)

- Trigger: quick run
- Logic: transform_json

Build steps (detailed)

462. Dashboard → New Workflow → Name it (use a short, URL-safe name if you plan to trigger it from Hub).
463. Add nodes exactly in the order shown. Connect them left-to-right.
464. For each node: open Inspector → set Mode → fill parameters → close Inspector.
465. Use Quick Run with the sample JSON. Keep your first run small and simple.
466. Inspect node-by-node outputs in Run History. Fix the first mismatch before moving forward.

Sample input JSON

```
{ "input": { "text": "Open https://example.com and notify me." } }
```

Expected output shape

```
{
  "deviceActions": [
    { "type": "open_url", "url": "https://example.com" },
    { "type": "notification", "title": "Done", "body": "Opened the URL." }
  ]
}
```

AI prompt guidance (if tutorial uses AI)

If your tutorial includes an AI node, make the model return strict JSON. This is the most common beginner failure mode.

```
Prompt snippet you can reuse:
Return ONLY valid JSON.
Do not include explanations or markdown.
JSON must match this schema exactly: { "key": "value" }
```

Hub run (optional)

467. Start Hub server and confirm LAN URL.
468. Pair iOS to Hub if pairing is enabled.
469. Trigger the workflow on Hub and verify it appears in Hub Run History.

Webhook test (optional)

```
curl -X POST "http://<hub-ip>:8787/trigger/<workflowName>" \
  -H "Content-Type: application/json" \
  -H "X-NodeSpark-Secret: <secret>" \
  -d '<PASTE INPUT JSON HERE>'
```

Troubleshooting

- JSON errors: validate braces and commas; use a JSON validator if needed.
- AI returns text: strengthen prompt with 'ONLY JSON' and show schema.
- Missing keys: align the input JSON keys with what the AI/transform expects.
- Hub unreachable: confirm Wi-Fi, correct IP/port, and server running.

7. JSON Practice Workbook

This workbook is designed to make you comfortable with JSON inside NodeSpark. Each exercise includes an input, a goal, and an example solution.

Exercise 1

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 2

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 3

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 4

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 5

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
```

```
    "tags": {{input.tags}}
  }
}
```

Exercise 6

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 7

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 8

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }
```

```
Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 9

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 10

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 11

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 12

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 13

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 14

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 15

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 16

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 17

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 18

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 19

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 20

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 21

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 22

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 23

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 24

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 25

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 26

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 27

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 28

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 29

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 30

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 31

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 32

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 33

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 34

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 35

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 36

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 37

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 38

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 39

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 40

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 41

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 42

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 43

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 44

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 45

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 46

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 47

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 48

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }
```

```
Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 49

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

Exercise 50

Goal: transform the input into the target shape using transform_json and (optionally) string cleanup nodes.

```
Input:
{ "input": { "first": "Jane", "last": "Doe", "city": "Chicago", "tags":
["a","b"] } }

Target:
{ "person": { "name": "Jane Doe", "location": "Chicago", "tags": ["a","b"]
} }

Example transform template:
{
  "person": {
    "name": "{{input.first}} {{input.last}}",
    "location": "{{input.city}}",
    "tags": {{input.tags}}
  }
}
```

8. Parameter Key Dictionary

This dictionary explains parameter keys detected in your builds. Many keys are shared across node modes; others are mode-specific.

__continueOnFail

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

__disabled

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

__notes

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

action

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionHTTPBody

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionHTTPHeaders

A dictionary of HTTP headers.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionHTTPMethod

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionHTTPURL

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).

- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionNotificationBody

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionNotificationTitle

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionType

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

actionURL

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

action_delay_seconds

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

action_mode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

action_type

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

aiAgentMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

aiInstruction

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

aiMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

aiWorkerID

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

aiWorkerId

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_agentMaxSteps

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_agentMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_apiKey_override

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_endpoint_override

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_model_override

An AI model selector or identifier.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_profileID

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_profileId

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_profile_id

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_prompt

A prompt template string for an AI node.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_provider_override

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_temperature_override

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_tool_appendLogEnabled
Whether a feature/node is enabled.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_tool_readVariableEnabled
Whether a feature/node is enabled.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_tool_subflowEnabled
Whether a feature/node is enabled.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_tool_subflowNames
A display name shown in UI.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_useCustomConfig
A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_workerId

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ai_worker_id

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

appevent_name

A display name shown in UI.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

auth_basic_password

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

auth_basic_username

A display name shown in UI.

Where you will see it:

- Node Inspector fields (most common).

- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

auth_bearer_token

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

auth_header

A dictionary of HTTP headers.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

body

HTTP request body (often JSON).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

bodyMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cal_end

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cal_location

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cal_start

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cal_title

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

clipboard_text

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cond_caseSensitive

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cond_left

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cond_op

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cond_right

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

conditionMatchText

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

conditionStopWhenFalse

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

conditionType

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

continueOnFail

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

continue_on_fail

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

cred_key

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

credentialID

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

credentialId

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

credentialService

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

credentials_profile_id

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

delay

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

delayMs

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

delaySeconds

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

delay_seconds

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

disabled

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

echo_template

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

exec_disabled

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

exec_onError

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

exec_retryDelayMs

Networking reliability setting (timeouts/retries).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

exec_retryMax

Networking reliability setting (timeouts/retries).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

exec_timeoutMs

Networking reliability setting (timeouts/retries).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

headersJson

A dictionary of HTTP headers.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

httpCredentialID

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_apiKeyAddTo

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_apiKeyHeader

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_apiKeyHeaderName

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_apiKeyQueryName

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_auth_mode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_body

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_body_json

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_follow_redirects

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_headers

A dictionary of HTTP headers.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_headers_json

A dictionary of HTTP headers.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_method

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_parse_json

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_service

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_timeout

Networking reliability setting (timeouts/retries).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

http_url

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).

- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

hubBaseURL

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

hub_worker_id

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

if_left

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

if_op

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

if_right

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

key

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

logicMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

logicType

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mail_body

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mail_subject

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mail_to

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

map_fields

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

map_fields_spec

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mappingsJson

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mergeJson

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

merge_json

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

merge_json_object

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

method

HTTP method (GET, POST, PUT, DELETE).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

mode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

ms

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

notify_body

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

notify_message

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

notify_title

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

onError

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

openurl

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

openurl_background

A URL field used by HTTP or open-url nodes.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

outputMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

overwrite

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

path

Key path in JSON (e.g., input.text).

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

promptProfileID

A prompt template string for an AI node.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

provider

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

reminder_notes

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

reminder_title

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_cron

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_day

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_days

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_kind

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_time

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_tz

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_weekday

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

schedule_weekends

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

seconds

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

service

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

service_key

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

share_text

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

shortcut_input

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

shortcut_name

A display name shown in UI.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

subflow_key

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

subflow_payload

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

subflow_workflowName

A display name shown in UI.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

targetPath

A JSON key path used to read/write values.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

transform_template

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerAppEventDescription

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerAppEventKind

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerMode

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerNotes

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerScheduleDescription

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerType

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

triggerWebhookDescription

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

trigger_enabled

Whether a feature/node is enabled.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

type

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

url

URL to fetch or open. Should include http:// or https://.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

value

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

valueType

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

varKey
A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

varValue
A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhookKey
A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhook_method
A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhook_path
A JSON key path used to read/write values.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhook_require_secret

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhook_secret_key

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

webhook_secret_on

A secret credential field. Store it in Connections/Keychain and reference it.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

worker_id

A parameter key used by one or more nodes. Confirm its behavior in the Inspector for the associated node mode.

Where you will see it:

- Node Inspector fields (most common).
- Transform templates and variable substitution contexts (where supported).
- Hub settings or webhook configuration (for server-related keys).

9. Screen Dictionary (View Inventory)

This section enumerates the screen/view structs detected in your builds. Names are developer-facing, but they map closely to what users see.

AboutView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

CommandPaletteView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ConnectionView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

CredentialsDefaultsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

CredentialsManagerView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

CredentialsProfilesManagerView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ElectricTitleView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ExecutionNodeTraceDetailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ExecutionTraceDetailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ExecutionTraceHistoryView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubAISettingsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.

- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubClientsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubDashboardView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubDeviceRunsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubDevicesView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubIntegrationPaywallView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubIntegrationsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubNodeView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubOnboardingView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubRemoteView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubRunDetailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubRunsCardView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.

- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubRunsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubScheduleEditorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubSchedulesView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubServerSheetView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubServerView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubSettingsContainerView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubSettingsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubWorkersView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

HubWorkflowsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

LogsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

MiniMapView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.

- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeCanvasView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeCardView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeDetailSheetView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeInspectorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeTestView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

NodeView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

OnboardingView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

PayloadBuilderView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

PayloadPresetEditorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

PromptProfileCardView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

PromptProfileEditorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.

- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

PromptProfileListView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

RootView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

RunHistoryDetailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

RunHistoryView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

TabRailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

TemplateCardView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

ToastView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowCanvasEditorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowCredentialMappingView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowCredentialsView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowEditorView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.

- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowIntegrationView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowListView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

WorkflowsDetailView

Purpose: A UI view in NodeSpark or NodeSparkHub. The exact layout depends on platform. If this is a primary screen, it is usually reachable from Dashboard, Settings, or a navigation flow from the editor.

User-oriented interpretation:

- What you can do here: configure, browse, run, inspect, or manage related items.
- What to look for: buttons, pickers, and inspector panels corresponding to node parameters.
- Troubleshooting: if something appears missing, confirm feature flags or subscription gating (if present).

10. Appendices (Full inventories)

10.1 NodeSpark AppStorage keys

- enableHaptics
- hasCompletedOnboarding
- iCloudSyncEnabled
- nodespark.hubAccessToken
- nodespark.hubBaseURL
- nodespark.hubIntegrationEnabled
- nodespark.hubPendingRunId
- nodespark.minimap.normX
- nodespark.minimap.normY
- showCanvasMiniMap

10.2 NodeSparkHub AppStorage keys

- HubServerView.consoleCollapsed
- HubServerView.consoleHeight
- hub.activePromptProfileID
- hub.autoStartServer
- hub.enableDeviceActionsOutput
- hub.enableRunHistory
- hub.keepMostRecentRuns
- hub.redactSecretsInLogs
- hub.serverPort
- hub.showAdvancedSettings
- hub.tlsEnabled
- hub.tlsP12Path
- hub.tlsPort

10.3 Hub endpoints

- /api/v1
- /api/v1/
- /clients
- /clients/ping
- /devices
- /devices/
- /devices/checkin
- /health
- /pair
- /result
- /run

- /runs
- /runs/
- /runs/latest
- /schedules
- /schedules/
- /status
- /tls/fingerprint
- /trace
- /trigger/
- /workers
- /workers/
- /workflows
- /workflows/