

COMPREHENSIVE CHEAT SHEET:

n8n Node JSON Definitions, Tools, Integrations & AI Agent Management

This guide provides a complete reference for n8n node JSON structures, integration nuances, and AI Agent configuration. It covers everything from chat triggers and sticky notes to advanced integrations with Pinecone, Airtable, PostgreSQL, Google Docs, Google Calendar, Telegram, HTTP APIs, SerpAPI, workflow tools, and more. It also includes best practices for AI Agent manipulation and addresses common property value issues when generating JSON through LLMs.

1. General n8n Node JSON Structure

Overview:

Every node is defined as a JSON object with standard keys applicable to all types — triggers, data processing, documentation, or integrations.

Key Top-Level Fields:

- **id:** Unique identifier (usually a UUID). Must be unique within a workflow.
- **name:** Human-readable label (e.g., "When Chat Message Received").
- **type:** Node functionality identifier (e.g., "@n8n/n8n-nodes-langchain.chatTrigger", "n8n-nodes-base.stickyNote").
- **typeVersion:** Schema version (commonly 1, 1.1, 1.7).
- **position:** [x, y] coordinates on the canvas.
- **parameters:** Node-specific configuration (actions, credentials, UI text, etc.).

Best Practices:

- Use descriptive names and organized layout.
 - Leverage n8n expressions (`={{ $json["field"] }}`) for dynamic content.
 - Validate JSON to prevent issues like improper quotes, trailing commas, or type mismatches.
-

2. Chat Trigger Nodes

(@n8n/n8n-nodes-langchain.chatTrigger)

Purpose: Entry point for conversation-based workflows via webhooks or embedded widgets.

Key Parameters:

- **webhookId:** Unique webhook identifier.
- **mode:** Typically `"webhook"`.
- **public:** Boolean for public endpoint access.
- **initialMessages:** Predefined instructions or greetings (supports Markdown).
- **options:** Advanced settings (responseMode, allowedOrigins, title/subtitle, file uploads, session persistence).

Nuances:

- Ensure unique webhook IDs.
 - Use dynamic expressions for runtime customization.
 - Type versions may differ in available parameters.
-

3. Sticky Note Nodes (n8n-nodes-base.stickyNote)

Purpose: Provide in-workflow documentation, annotations, or reminders.

Key Parameters:

- **content:** Markdown-supported text.
- **width & height:** Visual dimensions.
- **color:** Optional background color.

Usage Tips: Place near relevant nodes and maintain clear, concise formatting.

4. AI Agent Module

(@n8n/n8n-nodes-langchain.agent)

Purpose: Process user input, generate context-aware responses, and dynamically invoke tools.

Key Parameters:

- **text:** Primary input (e.g., `={{ $json.chatInput }}`).
- **options.systemMessage:** Defines agent role, memory, tool calls, and response formatting.
- **promptType:** Usually `"define"` to enforce system message rules.

Integration & Memory: Connects to `ai_tool`, `ai_memory`, and `ai_languageModel` ports. Often paired with memory nodes (`memoryBufferWindow`) for conversation history.

Nuances: Carefully craft `systemMessage` and validate output JSON. Test models for performance/cost.

Example:

```
{
  "id": "agent-1",
  "name": "AI Agent for Chat",
  "type": "@n8n/n8n-nodes-langchain.agent",
  "position": [100, 100],
  "parameters": {
    "text": "={{ $json.chatInput }}",
```

```
    "options": { "systemMessage": "=You are a helpful assistant." },
    "promptType": "define"
  },
  "typeVersion": 1.7
}
```

5. Memory Buffer Nodes

(**n8n-nodes-langchain.memoryBufferWindow**)

Purpose: Manage temporary conversation history for AI agents.

Key Parameters:

- **sessionKey:** Memory session identifier.
- **contextWindowLength:** Number of messages to retain.

Tips: Maintain consistent session keys and adjust window length based on conversation complexity.

6. Integration Tools & Operations

6.1 Pinecone: Vector DB for indexing, upserting, and querying. Ensure credentials and field mapping are correct.

6.2 Airtable: Manage records; exact field mapping required.

6.3 PostgreSQL: SQL queries for data manipulation; use parameterized queries.

6.4 Google Docs: Retrieve or update documents; map dynamic fields correctly.

6.5 Google Calendar: Create or retrieve events; validate date formats.

6.6 Telegram: Send/receive messages; dynamic chat IDs recommended.

6.7 HTTP Request: Make API calls; validate JSON payloads.

6.8 SerpAPI: Query search data; ensure proper credentials.

6.9 Workflow Tool: Trigger sub-workflows; validate input schema.

6.10 Gmail: Send emails; personalize using dynamic expressions.

6.11 Supabase: Interact with databases; secure API keys and field mapping.

6.12 Generic Nodes: Handle assignments, merges, splits, conditionals, and sub-workflows.

7. AI Agent & Memory Best Practices

- Craft detailed `systemMessage` for edge cases.
 - Use dynamic expressions to adapt to inputs.
 - Test full conversational flows to ensure data passes correctly.
 - Validate JSON outputs to avoid property errors.
-

8. Common JSON Issues with LLM-Generated Code

- **Quotation marks:** Use double quotes.
 - **Trailing commas:** Avoid them.
 - **Type mismatches:** Numbers/booleans should not be in quotes.
 - **Case sensitivity:** Match exact field names.
 - **Brackets/Braces:** Ensure proper structure.
 - **Validation:** Use JSON validators.
 - **LLM Prompting:** Specify “output valid JSON” and warn about common pitfalls.
-

9. Key Takeaways

- Chat Trigger Nodes: Entry points with dynamic greetings.
- Sticky Notes: In-workflow documentation for clarity.
- AI Agent Module: Integrates input, memory, and tool calls.
- Integration Tools: Each has unique requirements; ensure precise configuration.

- LLM-Generated JSON: Validate and format carefully.
- Well-documented structures + rigorous integrations = robust, maintainable workflows.

End of Comprehensive Cheat Sheet 