================================================================

COMPREHENSIVE CHEAT SHEET: n8n NODE JSON DEFINITIONS, ALL TOOLS, INTEGRATION NUANCES, & AI AGENT MANIPULATION

================================================================

This document combines a general overview of n8n node JSON structures with detailed sections on every tool encountered—from chat triggers and sticky notes to integrations with Pinecone, Airtable, PostgreSQL, Google Docs, Google Calendar, Telegram, HTTP APIs, SerpAPI, Workflow tools, and more. It also covers best practices for manipulating the AI Agent module and addresses common property value issues when generating JSON via LLMs.

────────────────────────────────────────────────────────────

1. GENERAL n8n NODE JSON STRUCTURE

────────────────────────────────────────────────────────────

Overview:

• Every node is defined as a JSON object with a common set of keys.

• These keys apply to all node types, whether they handle triggers, data processing, documentation, or external integrations.

Key Top-Level Fields:

– \*\*id:\*\* Unique identifier (typically a UUID). Must be unique within a workflow.

– \*\*name:\*\* Human-readable label (e.g., "When Chat Message Received", "Airtable - Update Records").

– \*\*type:\*\* Defines node functionality; examples:

• `"@n8n/n8n-nodes-langchain.chatTrigger"` for chat triggers.

• `"n8n-nodes-base.stickyNote"` for annotations.

• Other tools use specific type identifiers.

– \*\*typeVersion:\*\* Schema version (commonly 1, 1.1, 1.7, etc.). Higher versions may support additional features.

– \*\*position:\*\* An array `[x, y]` determining the node’s placement on the visual canvas.

– \*\*parameters:\*\* A nested object containing node-specific configurations (actions, credentials, UI text, etc.).

Best Practices:

• Use descriptive names and logical positions to maintain an organized workflow.

• Leverage n8n’s expression language (e.g., `={{ $json["field"] }}`) for dynamic content.

• Validate your JSON (e.g., with a linter) to catch property value issues like improper quotes, trailing commas, or type mismatches.

────────────────────────────────────────────────────────────

2. CHAT TRIGGER NODES (@n8n/n8n-nodes-langchain.chatTrigger)

────────────────────────────────────────────────────────────

Purpose:

• Listen for incoming chat messages via webhooks or embedded chat widgets.

• Act as the entry point for conversation-based workflows.

Key Parameters:

• \*\*webhookId:\*\* Unique identifier linking the node to its webhook endpoint.

• \*\*mode:\*\* Typically `"webhook"` to indicate event reception mode.

• \*\*public (boolean):\*\* Determines if the chat endpoint is open for public access. Use with secure `allowedOrigins`.

• \*\*initialMessages:\*\* Predefined greeting or instruction text (supports Markdown and dynamic expressions).

• \*\*options:\*\* Advanced settings including:

– \*\*responseMode:\*\* How responses are sent (e.g., `"responseNode"`).

– \*\*allowedOrigins:\*\* Domains permitted to access the chat endpoint.

– \*\*title/subtitle:\*\* Custom UI text.

– \*\*allowFileUploads:\*\* Enable/disable file uploads.

– \*\*loadPreviousSession:\*\* Manages session persistence (e.g., `"memory"`).

Nuances:

• Ensure each Chat Trigger’s webhookId is unique.

• Dynamic expressions in initialMessages allow runtime customization.

• Different typeVersions (1 vs. 1.1) may offer varying parameters.

────────────────────────────────────────────────────────────

3. STICKY NOTE NODES (n8n-nodes-base.stickyNote)

────────────────────────────────────────────────────────────

Purpose:

• Provide in-workflow documentation, annotations, or reminders.

• Do not affect data flow; purely for informational use.

Key Parameters:

• \*\*content:\*\* The display text (supports Markdown for headers, lists, code blocks, links, images).

• \*\*width & height:\*\* Set the visual dimensions on the canvas.

• \*\*color (optional):\*\* Numeric code to assign a background color for visual differentiation.

Usage Tips:

• Place sticky notes near related nodes to add context.

• Use clear and concise Markdown formatting.

• Update notes regularly as workflows evolve.

────────────────────────────────────────────────────────────

4. AI AGENT MODULE (n8n-nodes-langchain.agent)

────────────────────────────────────────────────────────────

Purpose:

• Process user input and generate context-aware, interactive responses.

• Dynamically invoke other tools based on conversation context and memory.

Key Parameters:

• \*\*text:\*\* Primary input, typically set dynamically (e.g., `={{ $json.chatInput }}`).

• \*\*options:\*\* Contains a detailed `systemMessage` that:

– Defines the AI agent’s role and behavior.

– Provides guidelines for memory management and tool invocation.

– Can include additional instructions for response formatting.

• \*\*promptType:\*\* Usually `"define"`, enforcing the system message rules.

Integration & Memory:

• Connects with subfunctions via ports like `ai\_tool`, `ai\_memory`, and `ai\_languageModel`.

• Often paired with memory nodes (e.g., `memoryBufferWindow`) to supply conversation history.

Nuances:

• Carefully craft the systemMessage to handle various scenarios.

• Validate output JSON for property value issues (see Section 6).

• Experiment with different models (e.g., "gpt-4o" vs. "gpt-4o-mini") for performance/cost balance.

Snippet Examples:

-----------------------------------------------------

\*Minimal AI Agent Configuration:\*

{

"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

}

-----------------------------------------------------

\*AI Agent with Calendar Integration:\*

{

"id": "agent-2",

"name": "Calendar AI Agent",

"type": "@n8n/n8n-nodes-langchain.agent",

"position": [200, 200],

"parameters": {

"text": "={{ $json.chatInput }}",

"options": { "systemMessage": "=You are a Google Calendar assistant. Ask for event details before creating an event." },

"promptType": "define"

},

"typeVersion": 1.7

}

-----------------------------------------------------

\*AI Agent with Memory Integration:\*

{

"id": "agent-3",

"name": "AI Agent with Memory",

"type": "@n8n/n8n-nodes-langchain.agent",

"position": [300, 300],

"parameters": {

"text": "={{ $json.chatInput }}",

"options": { "systemMessage": "=Use long-term memory to provide context-aware responses." },

"promptType": "define"

},

"typeVersion": 1.7

}

────────────────────────────────────────────────────────────

5. MEMORY BUFFER NODES (n8n-nodes-langchain.memoryBufferWindow)

────────────────────────────────────────────────────────────

Purpose:

• Manage temporary conversation history for the AI agent.

Key Parameters:

• \*\*sessionKey:\*\* Identifier for the memory session (can be dynamic).

• \*\*contextWindowLength:\*\* Number of messages to retain in the context window.

Usage Tips:

• Adjust contextWindowLength based on conversation complexity.

• Use consistent session keys to maintain memory continuity.

────────────────────────────────────────────────────────────

6. INTEGRATION TOOLS & THEIR OPERATIONS

────────────────────────────────────────────────────────────

This section details each external tool encountered and its specific nuances.

6.1. PINECONE

---------------

Purpose:

• Interface with Pinecone’s vector database for indexing, upserting, and querying vectors.

Key Operations:

• Creating/Updating Indexes – Define index names, dimensions, and metrics.

• Upserting Vectors – Map JSON fields to vector data; schema must match the index.

• Querying Vectors – Retrieve similar vectors based on query parameters.

Nuances:

• Ensure API credentials and field mappings are accurate.

• Use descriptive node names (e.g., "Pinecone - Upsert Vectors").

6.2. AIRTABLE

---------------

Purpose:

• Manage records within Airtable bases.

Key Operations:

• Reading, Creating, Updating, and Deleting Records.

• Field mapping: JSON keys must exactly match Airtable column names.

Nuances:

• Secure API keys via credentials.

• Use dynamic expressions to handle record data.

6.3. POSTGRESQL

---------------

Purpose:

• Execute SQL queries to manipulate data.

Key Operations:

• SELECT, INSERT, UPDATE, DELETE queries.

• Use parameterized queries to avoid SQL injection.

Nuances:

• Ensure SQL syntax is valid and dynamic values are correctly inserted.

• Use nodes like postgresTool for JSON manipulation tasks.

6.4. GOOGLE DOCS

---------------

Purpose:

• Retrieve or update Google Docs for storing long-term memory or notes.

Key Operations:

• “get” for retrieval and “update” for inserting content.

• Use actionsUi (in googleDocsTool) to define JSON payloads for operations.

Nuances:

• Link correct OAuth credentials.

• Map dynamic fields (e.g., dates, memory content) properly.

6.5. GOOGLE CALENDAR

---------------

Purpose:

• Retrieve or create events in Google Calendar.

Key Operations:

• Get Events: Use date range filters with dynamic expressions.

• Create Events: Specify start, end, summary, description, and additional fields (attendees, conference data).

Nuances:

• Validate date formats (YYYY-MM-DD HH:mm:ss).

• Ensure calendar field mappings match the Google Calendar API.

6.6. TELEGRAM

---------------

Purpose:

• Manage messaging through Telegram.

Key Node Types:

• telegramTrigger: For receiving messages.

• telegram: For sending text replies.

• telegramTool: For sending files/documents.

Key Parameters:

• webhookId (for triggers), chatId, text, file (URL), and additionalFields (e.g., parse\_mode).

Nuances:

• Use dynamic expressions to retrieve chat IDs.

• Test with sandbox credentials when possible.

6.7. HTTP REQUEST

---------------

Purpose:

• Make generic HTTP API calls (e.g., for DALL-E image generation).

Key Parameters:

• url, method, sendBody, sendHeaders.

• authentication: Predefined or generic credentials.

Nuances:

• Ensure JSON bodies are valid.

• Use this node for calling external APIs such as DALL-E by setting the model in the payload.

6.8. SERPAPI TOOL

---------------

Purpose:

• Query search engine data via SerpAPI.

Key Parameters:

• options: Additional query settings.

Nuances:

• Link proper SerpAPI credentials.

• Formulate dynamic search queries based on user input.

6.9. WORKFLOW TOOL

---------------

Purpose:

• Trigger or execute separate workflows (e.g., task creation).

Key Parameters:

• name, workflowId, schemaType, and inputSchema.

Nuances:

• Validate that the input JSON schema matches the target workflow.

• Use for automating sub-workflows.

6.10. GMAIL TOOL

---------------

Purpose:

• Send emails via Gmail.

Key Parameters:

• sendTo, subject, message, additionalFields.

Nuances:

• Personalize emails with dynamic expressions.

• Ensure OAuth credentials are correctly set.

6.11. SUPABASE

---------------

Purpose:

• Interact with Supabase databases.

Key Parameters:

• tableId, fieldsUi (mapping JSON data to table fields).

Nuances:

• Ensure the data schema aligns with your Supabase table.

• Secure API keys via credentials.

6.12. GENERIC & AUXILIARY NODES

---------------

Purpose:

• Handle general operations such as value assignments, merging data, splitting outputs, conditionals, and triggering sub-workflows.

Key Examples:

• Set Nodes: For assigning values (e.g., system prompts).

• Merge, Aggregate, Split Out: For data orchestration.

• If Nodes: For conditional logic.

• Execute Workflow Trigger & Respond To Webhook: For control flow.

Nuances:

• Keep configurations clean and validate dynamic expressions.

────────────────────────────────────────────────────────────

7. ADDITIONAL NOTES ON AI AGENT & MEMORY MANIPULATION

────────────────────────────────────────────────────────────

• The AI Agent module orchestrates intelligent responses and tool calls.

• Key features include:

– Processing user input via the "text" parameter.

– Guiding behavior with a detailed "options.systemMessage" (which can include memory, tool-calling rules, and fallback responses).

– Connecting to memory nodes (e.g., memoryBufferWindow) for context retention.

• Best Practices:

– Craft comprehensive system messages that cover edge cases.

– Use dynamic expressions to adapt to varying inputs.

– Test the full conversational flow to ensure dynamic values (e.g., event details) pass correctly between nodes.

– Validate JSON outputs to avoid common property value issues (see Section 6).

────────────────────────────────────────────────────────────

8. COMMON PROPERTY VALUE ISSUES WITH LLM-GENERATED JSON

────────────────────────────────────────────────────────────

• \*\*Quotation Marks:\*\*

– Use double quotes for property names and string values.

• \*\*Trailing Commas:\*\*

– Avoid trailing commas after the last element in objects or arrays.

• \*\*Data Type Mismatches:\*\*

– Do not wrap numeric or boolean values in quotes.

• \*\*Case Sensitivity:\*\*

– Ensure property names match expected casing.

• \*\*Brackets/Braces:\*\*

– Confirm all JSON structures are properly opened and closed.

• \*\*Validation:\*\*

– Run your JSON through validators to catch syntax errors.

• \*\*LLM Prompting:\*\*

– When instructing LLMs to generate JSON, specify “output valid JSON” and warn about these common pitfalls.

────────────────────────────────────────────────────────────

9. FINAL TAKEAWAYS

────────────────────────────────────────────────────────────

• \*\*Chat Trigger Nodes:\*\*

– Act as entry points for conversation-based workflows with nuanced parameters for public access and dynamic greetings.

• \*\*Sticky Note Nodes:\*\*

– Serve as in-workflow documentation and are essential for clarity in complex workflows.

• \*\*AI Agent Module:\*\*

– Integrates user input, dynamic memory, and tool calls to generate context-aware responses.

• \*\*Integration Tools:\*\*

– Each external tool (Pinecone, Airtable, PostgreSQL, Google Docs/Calendar, Telegram, HTTP Request, SerpAPI, Workflow Tool, Gmail, Supabase) has specific configuration needs and requires precise field mappings.

• \*\*LLM-Generated JSON:\*\*

– Address common property value issues by careful formatting and validation.

• \*\*Overall:\*\*

– Combining well-documented node structures with rigorous integration setups leads to robust, maintainable workflows.

================================================================

END OF COMPREHENSIVE CHEAT SHEET

================================================================