DNOTIFIER SDK CHEAT SHEET

Everything you need to build realtime apps and AI agents on DNotifier

Copy-paste reference for the JavaScript / TypeScript SDK. Python and Dart use the same methods: Python in snake_case (`send_ai`, `add_document`), Dart in camelCase.

STEP 1	STEP 2	STEP 3	STEP 4	STEP 5
Create an app, copy keys	Install the SDK	Connect	Send messages or ask AI	Add agents, tools, workflows

01 SET UP YOUR APP

Sign up with Google at app.dnotifier.com, then **Apps → Create app** and copy the **appId** and **secret**.

```
# .env  (add .env to .gitignore, never commit it)
DNOTIFIER_APP_ID=your-app-id
DNOTIFIER_SECRET=your-secret
```

Add your model keys in **AI Studio → Providers** (OpenAI, Anthropic, Gemini...).

02 INSTALL

JavaScript / TypeScript (Node.js also needs **ws** for WebSocket)

```
npm install @dnotifier-realtime/dnotifier
npm install ws
```

Python

```
pip install dnotifier
```

Dart / Flutter

```
flutter pub add dnotifier
```

03 CONNECT

```
import { DNotifier } from "@dnotifier-realtime/dnotifier";

const notifier = new DNotifier({
  appId: process.env.DNOTIFIER_APP_ID,
  secret: process.env.DNOTIFIER_SECRET,
  transport: "http",      // or "ws"
  userId: "user-123",
});

await notifier.connect();
```

Check plan limits and whether AI is on

```
notifier.getPlanLimits();
notifier.aiEnabled;
```

Node.js with **ws**: pass the WebSocket class

```
import WebSocket from "ws";   // then:
WebSocketImpl: WebSocket
```

04 PICK A TRANSPORT

`"ws"`	`"http"`
Live chat, realtime messages, binary data, remote agents, waiting for replies	sendAI, search, knowledge-base documents, runWorkflow. Each call returns its answer.

Common setup: the chat client on **ws**, the AI backend on **http**.

05 REALTIME MESSAGING

Send to one user (direct, by userId)

```
await notifier.send({
  senderId: "alice", receiverId: "bob",
  data: { type: "text", text: "Hi Bob" },
});
```

Send to many users

```
receiverIds: ["bob", "carol", "dave"]
```

Receive

```
onMessage: (msg) => {
  msg.metadata.sender;      // who sent it
  msg.payload.toJSON();     // { type, text }
}
```

Images, audio, files

```
data: { type: "image", content: bytes,
fileName, mimeType }
```

Raw bytes (ws only). Read with `msg.payload.raw()`

```
notifier.sendBinary({ senderId, receiverIds,
buffer, type: "binary" });
```

06 CHAT HISTORY

History is saved by default. Skip it per message:

```
saveHistory: false
```

Load a conversation

```
await notifier.fetchChatHistory({ senderId:
"alice", receiverIds: ["bob"] });
```

Delete a message (id from `msg.metadata.id`)

```
await notifier.deleteChatHistoryMessage({
senderId, messageId });
```

07 ASK THE AI

```
const res = await notifier.sendAI({
  senderId: "user-123",
  message: { text: "Summarize this ticket." },
});
const answer = res?.data?.content;
```

Chat-style with a system prompt

```
message: { messages: [
  { role: "system", content: "You are a
billing expert." },
  { role: "user", content: "I was charged
twice." },
] }
```

Choose provider and model per call

```
provider: "open_ai", model: "gpt-4o"
```

Keep context across turns

```
const sessionId = res?.metadata?.packet?.id;
await notifier.sendAI({ senderId, sessionId,
message: { text: "And next?" } });
```

AI history

```
notifier.fetchAIHistory({ senderId });
notifier.deleteAIHistoryMessage({ senderId,
messageId });
```

08 KNOWLEDGE BASE (RAG)

Add a document. No vector database to run.

```
await notifier.addDocument({
  senderId, recordId: "refund-policy",
  content: "Refunds take 5 business days.",
  type: "text", metadata: { source: "help-
center" },
});
```

Update, read, list, delete

```
updateDocument()  getDocument()
listDocuments({ limit, offset })
deleteDocument()
```

Semantic search

```
const hits = await notifier.search({
  senderId, query: "how long do refunds take",
  limit: 5, minSimilarity: 0.7,
filterbySource: "help-center",
});
```

Answer from your docs: pass **hits** as context in a system message to **sendAI**.

09 AGENTS AND TOOLS

Register a tool

```
const add = notifier.tools.register({
  name: "math.add", description: "Add two
numbers",
  inputSchema: { type: "object", properties: {
    a: { type: "number" }, b: { type: "number"
} } },
  async execute(input) { return { sum: input.a
+ input.b }; },
});
```

Define an agent that can use it

```
const agent = DNotifier.defineAgent({
  name: "math-assistant",
  instructions: "Use math.add for sums.",
  provider: "open_ai", model: "gpt-4o",
  tools: [add],
  permissions: { "math.add": ["execute"] },
});
```

Run it

```
const r = await notifier.runAgent(agent, {
input: "21 + 21?", maxToolRounds: 6 });
r.output;  r.finishReason;  r.executionId;
```

Enforce permissions on the server: `permissionEnforcement: "backend"` in the constructor.

10 MCP SERVERS

```
const mermaid = await notifier.mcp.connect({
  id: "mermaid", name: "Agentic Mermaid",
  transport: "http",
  endpoint: "https://agentic-mermaid.dev/mcp",
});
const tools = mermaid.tools();  //
"mermaid.verify", …
```

Give them to an agent with **tools: tools**. Tool names are prefixed with the server id.

11 MULTI-AGENT WORKFLOWS

```
const flow = new DNotifier.Workflow({
  name: "support-router",
  observability: true,
  async entry(ctx) {
    const intent = await ctx.runAgent("intent-
agent");
    return ctx.runAgent("general-agent", {
input: ctx.input });
  },
}).registerAgents({
  "intent-agent": intentAgent,
  "general-agent": generalAgent,
});

const run = await notifier.runWorkflow({
workflow: flow, input: "How do refunds work?"
});
```

Inside **ctx**

`ctx.runAgents([…])`	run agents in parallel
`ctx.state`	share data between agents
`ctx.sendAI / ctx.search`	AI and RAG inside a step
`ctx.waitForMessage()`	pause for a human reply (ws)
`ctx.recordStep({ label })`	add your own step to the graph

Custom agent logic

```
DNotifier.defineAgent({ name: "writer", async
run(ctx) {
  return ctx.sendAI({ message: { text:
ctx.input } });
} });
```

Agent on another server: register `{ receiverId: "svc-agent-writer" }`, and on that host call `workflow.listen(notifier)` (ws).

12 MODEL PROVIDERS

`"open_ai"`	`"anthropic"`
`"google_gemini"`	`"hugging_face"`
`"perplexity"`	`"azure_ai"`
`"aws_bedrock"`	`"ollama"`

Set a default on the agent; override per call with **provider** and **model**.

13 SEE WHAT HAPPENED

`observability: true`	execution graph in Dashboard → Workflows
`logs: true`	AI sessions in Dashboard → Logs
`label: "Intent check"`	name a step in the graph
`onAgentEvent`	stream tool calls in your code

Usage and quotas: Dashboard → **Usage** and **Plan & limits**.

14 SAME CALLS IN PYTHON AND DART

JS / Dart	Python	Note
`sendAI`	`send_ai`	sender_id=
`addDocument`	`add_document`	doc_type=
`runWorkflow`	`run_workflow`	input_data=
`ctx.runAgent`	`ctx.run_agent`	async
`defineAgent`	`define_agent`	import from dnotifier

AI features need a paid plan. Check `notifier.aiEnabled` before calling AI.

Docs GitHub dnotifier.com