DNOTIFIER

QUICKSTART CHEAT SHEET

From zero to your first AI agent in 8 steps

Follow the steps in order. Each one has the code to paste and what you should see when it works. Examples use JavaScript; the Python and Dart SDKs have the same methods.

01 App & keys
02 Install
03 Connect
04 First message
05 Ask the AI
06 Your docs (RAG)
07 Agent + tool
08 Workflow

01

Create your app and keys

1. Go to app.dnotifier.com and sign up with Google.
2. Open **Apps → Create app**. Copy the **appId** and **secret**.
3. Open **AI Studio → Providers** and add your model key (OpenAI, Anthropic, Gemini...).

AI features need a paid plan. Realtime messaging works on every plan.

```env
# keep this file out of git (.gitignore)
DNOTIFIER_APP_ID=your-app-id
DNOTIFIER_SECRET=your-secret
```

02

Install the SDK

One package per language. In Node.js, also install **ws** so the SDK can open a WebSocket.

`npm · JS/TS` `pip · Python` `pub · Dart/Flutter`

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

# Python
pip install dnotifier

# Dart / Flutter
flutter pub add dnotifier
```

03

Connect

Every user or service gets a **userId**. Messages are delivered to that id directly.

Use **"ws"** for live messages and **"http"** for AI calls, search and workflows. You can run both in one app.

```connect.js
import WebSocket from "ws";
import { DNotifier } from "@dnotifier-realtime/dnotifier";

const notifier = new DNotifier({
  appId: process.env.DNOTIFIER_APP_ID,
  secret: process.env.DNOTIFIER_SECRET,
  transport: "ws",
  userId: "user-alice",
  WebSocketImpl: WebSocket,
  onConnected: () => console.log("Connected"),
  onMessage: (msg) => console.log(msg.payload.toJSON()),
});

await notifier.connect();
console.log("isConnected:", notifier.isConnected);
```

```
Connected
isConnected: true
```

04

Send your first realtime message

Run a second script as **user-bob** with the same connect code. Alice sends, Bob's **onMessage** fires.

Need a group? Use `receiverIds: ["bob", "carol"]`. Offline users don't get live delivery, but history is saved by default.

send.js

```
await notifier.send({
  senderId: "user-alice",
  receiverId: "user-bob",
  data: { type: "text", text: "Hello from DNotifier!" },
});
```

```
Bob received: { type: 'text', text: 'Hello from DNotifier!' }
```

05

Ask the AI

Switch to **"http"** for AI so each call returns its answer. Pick any of 8 providers per call, or leave them out to use your project default.

Multi-turn chat: pass back `sessionId` from the first response's `metadata.packet.id`.

ask.js

```
if (!notifier.aiEnabled) throw new Error("AI is not on for
this plan");

const res = await notifier.sendAI({
  senderId: "user-123",
  provider: "open_ai", model: "gpt-4o",   // optional
  message: { text: "Summarize our refund policy in two
sentences." },
});

console.log("AI:", res?.data?.content);
```

06

Teach it your docs (RAG)

Add documents to the built-in knowledge base, search them by meaning, then answer from the results. No vector database to host.

Also: `updateDocument`, `getDocument`, `listDocuments`, `deleteDocument`.

rag.js

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

const hits = await notifier.search({
  senderId: "user-123", query: question,
  limit: 3, minSimilarity: 0.5,
});

const res = await notifier.sendAI({
  senderId: "user-123",
  message: { messages: [
    { role: "system", content: `Answer only from:
${JSON.stringify(hits)}` },
    { role: "user", content: question },
  ] },
});
```

07

Give an agent a tool

Register a function as a tool, give it to an agent, and say who may run it. The agent decides when to call it.

Using an MCP server? `notifier.mcp.connect(…)` then pass its `.tools()` to the agent.

agent.js

```
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 }; },
});

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

const r = await notifier.runAgent(agent, { input: "What is 21
+ 21?" });
console.log(r.output, r.finishReason);
```

```
… "stop"
```

08

Orchestrate agents in a workflow

A workflow runs named agents in order or in parallel and shares state between them. Turn on **observability** to see every step in Dashboard → Workflows.

Parallel: `ctx.runAgents([…])`. Shared data: `ctx.state`. Human approval: `ctx.waitForMessage()` on ws.

workflow.js

```
// researcher, writer = DNotifier.defineAgent({ name, async
run(ctx) {…} })
const flow = new DNotifier.Workflow({
  name: "research-then-write",
  observability: true,
  async entry(ctx) {
    const notes = await ctx.runAgent("researcher");
    ctx.state.notes = notes;
    return ctx.runAgent("writer", { input: notes });
  },
}).registerAgents({ researcher, writer });

const run = await notifier.runWorkflow({ workflow: flow,
input: "Agent memory" });
console.log(run.result, run.executionId);
```

Keep going

Chat history

Load or delete past messages in a thread.

`fetchChatHistory({ senderId, receiverIds })`

Files and binary

Send images, audio, documents or raw bytes.

`sendBinary({ senderId, receiverIds, buffer })`

MCP servers

Connect remote tools over HTTP and hand them to agents.

`notifier.mcp.connect({ id, endpoint })`

Human in the loop

Pause a workflow until a person replies.

`ctx.waitForMessage({ timeoutMs })`

Remote agents

Run an agent on another server; the workflow calls it by id.

`{ receiverId: "svc-agent-writer" }`

8 model providers

OpenAI, Anthropic, Gemini, Hugging Face, Perplexity, Azure AI, AWS Bedrock, Ollama.

`provider: "aws_bedrock"`

Stuck? Check these first

`Not connected`
Call methods after **await connect()** finishes.

`AI is not enabled`
Check **notifier.aiEnabled** and your plan.

`receiverIds required`
Pass a non-empty array of userIds.

`Message not received`
Is the receiver online and using the same **appId**?

Get your API keys: app.dnotifier.com · Docs: dnotifier.gitbook.io/product-docs · Examples: github.com/dnotifier-ai