readme.md for @push.rocks/smartagent

A lightweight agentic loop built on **Vercel AI SDK v6** via `@push.rocks/smartai`. Register tools, get a model, call `runAgent()` — done. 🤖

Install

```
pnpm install @push.rocks/smartagent
```

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

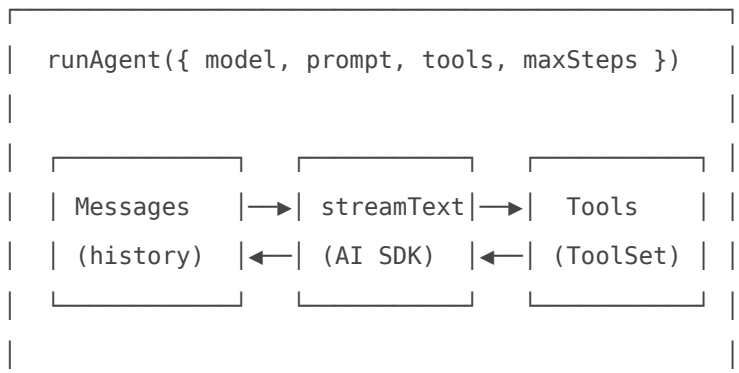
Overview

`@push.rocks/smartagent` wraps the AI SDK's `streamText` with `stopWhen: stepCountIs(n)` for **parallel multi-step tool execution**. No classes to instantiate, no lifecycle to manage — just one async function:

```
import { runAgent, tool, z } from '@push.rocks/smartagent';
import { getModel } from '@push.rocks/smartai';

const model = getModel({
  provider: 'anthropic',
  model: 'claude-sonnet-4-5-20250929',
```

```
  apiKey: process.env.ANTHROPIC_TOKEN,
});

const result = await runAgent({
  model,
  prompt: 'What is 7 + 35?',
  system: 'You are a helpful assistant. Use tools when asked.',
  tools: {
    calculator: tool({
      description: 'Perform arithmetic',
      inputSchema: z.object({
        operation: z.enum(['add', 'subtract', 'multiply', 'divide']),
        a: z.number(),
        b: z.number(),
      }),
      execute: async ({ operation, a, b }) => {
        const ops = { add: a + b, subtract: a - b, multiply: a * b, divide: a / b };
        return String(ops[operation]);
      },
    }),
  },
  maxSteps: 10,
});

console.log(result.text);    // "7 + 35 = 42"
console.log(result.steps);   // number of agentic steps taken
console.log(result.usage);   // { promptTokens, completionTokens, totalTokens }
```

Architecture



```
│  stopWhen: stepCountIs(maxSteps)                │
│  + retry with backoff on 429/529/503           │
│  + context overflow detection & recovery       │
│  + tool call repair (case-insensitive matching) │
└─────────────────────────────────────────────────┘
```

Key features:

- □□ **Multi-step agentic loop** — the model calls tools, sees results, and continues reasoning until done
- ⚡ **Parallel tool execution** — multiple tool calls in a single step are executed concurrently
- □□ **Auto-retry with backoff** — handles 429/529/503 errors with header-aware retry delays
- □□ **Tool call repair** — case-insensitive name matching + invalid tool sink prevents crashes
- □□ **Token streaming** — `onToken` and `onToolCall` callbacks for real-time progress
- □□ **Context overflow handling** — detects overflow and invokes your `onContextOverflow` callback

Core API

`runAgent(options): Promise<IAgentRunResult>`

The single entry point. Options:

Option	Type	Default	Description
`model`	`LanguageModelV3`	*required*	Model from `@push.rocks/smartai`'s `getModel()`
`prompt`	`string`	*required*	The user's task/question
`system`	`string`	`undefined`	System prompt
`tools`	`ToolSet`	`{}`	Tools the agent can call
`maxSteps`	`number`	`20`	Max agentic steps before stopping
`messages`	`ModelMessage[]`	`[]`	Conversation history (for multi-turn)

Option	Type	Default	Description
`maxRetries`	`number`	`5`	Max retries on rate-limit/server errors
`onToken`	`(delta: string) => void`	—	Streaming token callback
`onToolCall`	`(name: string) => void`	—	Called when a tool is invoked
`onContextOverflow`	`(messages) => messages`	—	Handle context overflow (e.g., compact messages)

`IAgentRunResult`

```
interface IAgentRunResult {
  text: string;              // Final response text
  finishReason: string;      // 'stop', 'tool-calls', 'length', etc.
  steps: number;             // Number of agentic steps taken
  messages: ModelMessage[];  // Full conversation for multi-turn
  usage: {
    promptTokens: number;
    completionTokens: number;
    totalTokens: number;
  };
}
```

Defining Tools □□

Tools use Vercel AI SDK's `tool()` helper with Zod schemas:

```
import { tool, z } from '@push.rocks/smartagent';

const myTool = tool({
  description: 'Describe what this tool does',
  inputSchema: z.object({
    param1: z.string().describe('What this parameter is for'),
    param2: z.number().optional(),
  }),
  execute: async ({ param1, param2 }) => {
    // Do work, return a string
```

```
    return `Result: ${param1}`;
  },
});
```

Pass tools as a flat object to `runAgent()`:

```
await runAgent({
  model,
  prompt: 'Do the thing',
  tools: { myTool, anotherTool },
  maxSteps: 10,
});
```

ToolRegistry

A lightweight helper for collecting tools:

```
import { ToolRegistry, tool, z } from '@push.rocks/smartagent';

const registry = new ToolRegistry();

registry.register('random_number', tool({
  description: 'Generate a random integer between min and max',
  inputSchema: z.object({
    min: z.number(),
    max: z.number(),
  }),
  execute: async ({ min, max }) => {
    return String(Math.floor(Math.random() * (max - min + 1)) + min);
  },
}));

registry.register('is_even', tool({
  description: 'Check if a number is even',
  inputSchema: z.object({ number: z.number() }),
  execute: async ({ number: n }) => n % 2 === 0 ? 'Yes' : 'No',
}));
```

```
const result = await runAgent({
  model,
  prompt: 'Generate a random number and tell me if it is even',
  tools: registry.getTools(),
  maxSteps: 10,
});
```

Built-in Tool Factories □□

Import from the `@push.rocks/smartagent/tools` subpath:

```
import { filesystemTool, shellTool, httpTool, jsonTool } from '@push.rocks/smartagent/tools';
```

`filesystemTool(options?)`

Returns: `read_file`, `write_file`, `list_directory`, `delete_file`

```
const tools = filesystemTool({ rootDir: '/home/user/workspace' });

await runAgent({
  model,
  prompt: 'Create a file called hello.txt with "Hello World"',
  tools,
  maxSteps: 5,
});
```

Options:

- `rootDir` — restrict all file operations to this directory. Paths outside it throw `Access denied` .

`shellTool(options?)`

Returns: `run_command`

```
const tools = shellTool({ cwd: '/tmp', allowedCommands: ['ls', 'echo', 'cat'] });

await runAgent({
```

```
  model,
  prompt: 'List all files in /tmp',
  tools,
  maxSteps: 5,
});
```

Options:

- `cwd` — working directory for commands
- `allowedCommands` — whitelist of allowed commands (if set, others are rejected)

`httpTool()`

Returns: `http_get`, `http_post`

```
const tools = httpTool();

await runAgent({
  model,
  prompt: 'Fetch the data from https://api.example.com/status',
  tools,
  maxSteps: 5,
});
```

`jsonTool()`

Returns: `json_validate`, `json_transform`

```
const tools = jsonTool();

// Direct usage:
const result = await tools.json_validate.execute({
  jsonString: '{"name":"test","value":42}',
  requiredFields: ['name', 'value'],
});
// → "Valid JSON: object with 2 keys"
```

Streaming & Callbacks □□

Monitor the agent in real-time:

```
const result = await runAgent({
  model,
  prompt: 'Analyze this data...',
  tools,
  maxSteps: 10,

  // Token-by-token streaming
  onToken: (delta) => process.stdout.write(delta),

  // Tool call notifications
  onToolCall: (toolName) => console.log(`\n□□Calling: ${toolName}`),
});
```

Context Overflow Handling □□

For long-running agents that might exceed the model's context window, use the compaction subpath:

```
import { runAgent } from '@push.rocks/smartagent';
import { compactMessages } from '@push.rocks/smartagent/compaction';

const result = await runAgent({
  model,
  prompt: 'Process all 500 files...',
  tools,
  maxSteps: 100,

  onContextOverflow: async (messages) => {
    // Summarize the conversation to free up context space
    return await compactMessages(model, messages);
  },
});
```

Output Truncation ✂

Prevent large tool outputs from consuming too much context:

```
import { truncateOutput } from '@push.rocks/smartagent';

const { content, truncated, notice } = truncateOutput(hugeOutput, {
  maxLines: 2000,   // default
  maxBytes: 50_000, // default
});
```

The built-in tool factories use `truncateOutput` internally.

Multi-Turn Conversations □□

Pass the returned `messages` back for multi-turn interactions:

```
// First turn
const turn1 = await runAgent({
  model,
  prompt: 'Create a project structure',
  tools,
  maxSteps: 10,
});

// Second turn — continues the conversation
const turn2 = await runAgent({
  model,
  prompt: 'Now add a README to the project',
  tools,
  maxSteps: 10,
  messages: turn1.messages, // pass history
});
```

Exports

Main (`@push.rocks/smartagent`)

Export	Type	Description
`runAgent`	function	Core agentic loop
`ToolRegistry`	class	Tool collection helper
`truncateOutput`	function	Output truncation utility
`ContextOverflowError`	class	Error type for context overflow
`tool`	function	Re-exported from `@push.rocks/smartai`
`z`	object	Re-exported Zod for schema definitions
`stepCountIs`	function	Re-exported from AI SDK
`jsonSchema`	function	Re-exported from `@push.rocks/smartai`

Tools (`@push.rocks/smartagent/tools`)

Export	Type	Description
`filesystemTool`	factory	File operations (read, write, list, delete)
`shellTool`	factory	Shell command execution
`httpTool`	factory	HTTP GET/POST requests
`jsonTool`	factory	JSON validation and transformation

Compaction (`@push.rocks/smartagent/compaction`)

Export	Type	Description
`compactMessages`	function	Summarize message history to free context

Dependencies

- `@push.rocks/smartai` — Provider registry, `getModel()`, re-exports `tool`/`jsonSchema`
- `ai` v6 — Vercel AI SDK (`streamText`, `stepCountIs`, `ModelMessage`)
- `zod` — Tool input schema definitions

- `@push.rocks/smartfs` — Filesystem tool implementation
- `@push.rocks/smartshell` — Shell tool implementation
- `@push.rocks/smartrequest` — HTTP tool implementation

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

Revision #3
Created 2026-03-28 11:13:41 UTC by foss.global Team
Updated 2026-03-28 12:20:26 UTC by foss.global Team