LangChain Docs

Core components › **Tools**

Core components

Tools

Copy page

Tools extend what **agents** can do—letting them fetch real-time data, execute code, query external databases, and take actions in the world.

Under the hood, tools are callable functions with well-defined inputs and outputs that get passed to a **chat model**. The model decides when to invoke a tool based on the conversation context, and what input arguments to provide.

For details on how models handle tool calls, see **Tool calling**.

Create tools

Basic tool definition

The simplest way to create a tool is by importing the `tool` function from the `langchain` package. You can use **zod** to define the tool's input schema:

LangChain Docs

```
    ({ query, limit }) => `Found ${limit} results for '${query}'`,
    {
      name: "search_database",
      description: "Search the customer database for records matching the query."
      schema: z.object({
        query: z.string().describe("Search terms to look for"),
        limit: z.number().describe("Maximum number of results to return"),
      }),
    }
  );
```

Server-side tool use

Some chat models (e.g., **OpenAI**, **Anthropic**, and **Gemini**) feature **built-in tools** that are executed server-side, such as web search and code interpreters. Refer to the **provider overview** to learn how to access these tools with your specific chat model.

Accessing context

Why this matters: Tools are most powerful when they can access agent state, runtime context, and long-term memory. This enables tools to make context-aware decisions, personalize responses, and maintain information across conversations.

The runtime context provides a structured way to supply runtime data, such as DB connections, user IDs, or config, into your tools. This avoids global state and keeps tools testable and reusable.

Context

Tools can access an agent's runtime context through the `config` parameter:

LangChain Docs

Core components › **Tools**

LangChain Docs

```
const getUserName = tool(
  (_, config) => {
    return config.context.user_name
  },
  {
    name: "get_user_name",
    description: "Get the user's name.",
    schema: z.object({}),
  }
);

const contextSchema = z.object({
  user_name: z.string(),
});

const agent = createAgent({
  model: new ChatOpenAI({ model: "gpt-4o" }),
  tools: [getUserName],
  contextSchema,
});

const result = await agent.invoke(
  {
    messages: [{ role: "user", content: "What is my name?" }]
  },
  {
    context: { user_name: "John Smith" }
  }
);
```

Memory (Store)

ss persistent data across conversations using the store. The store is accessed via

LangChain Docs

```
import { createAgent, tool } from "langchain";
import { InMemoryStore } from "@langchain/langgraph";
import { ChatOpenAI } from "@langchain/openai";

const store = new InMemoryStore();

```

··· See all 70 lines

Stream writer

Stream custom updates from tools as they execute using `config.streamWriter` . This is useful for providing real-time feedback to users about what a tool is doing.

```
  ({ city }, config: ToolRuntime) => {
    const writer = config.writer;

    // Stream custom updates as the tool executes
    if (writer) {
      writer(`Looking up data for city: ${city}`);
      writer(`Acquired data for city: ${city}`);
    }

    return `It's always sunny in ${city}!`;
  },
  {
    name: "get_weather",
    description: "Get weather for a given city.",
    schema: z.object({
      city: z.string(),
    }),
  }
);
```

LangChain Docs

Core components › **Tools**

LangChain Docs

Resources

Forum

Changelog

LangChain Academy

Trust Center

Company

About

Careers

Blog

LangChain Docs

Core components › **Tools**