days of central knowledge 2026

From OpenAPI to Copilot – Auto-Generating MCP Tools for VS Code

4PS & COSMO CONSULT
21.05.2026

days of central knowledge 2026

Markus Lippert

Business
Platform Lead at COSMO CONSULT
Product Manager for COSMO Alpaca

working mainly on DevOps, developer productivity & tooling around it.

Community
Blogs at lippertmarkus.com

Find me and let's connect on LinkedIn and Bluesky

COSMOCONSULT

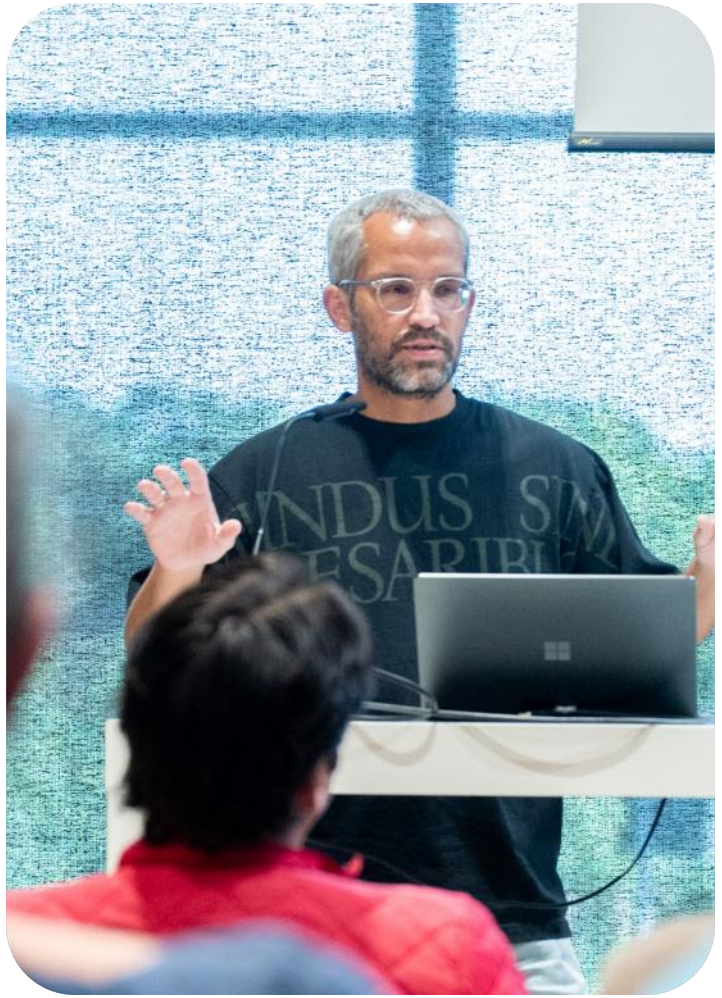
days of central knowledge 2026

Tobias Fenster

Business
General Manager at 4PS Germany
Chief Engineer at Hilti

Community
Microsoft MVP for Business Central and Azure; Regional Director
Docker Captain
Blog and podcast “Window on Technology” at tobiasfenster.io

Find me and let’s connect on LinkedIn and Bluesky



4PS BY HILTI

Agenda

- Intro & Motivation
- Basics of OpenAPI & MCP
- Live Demo
- Tips & Tricks
- Wrap-Up

Why do we need this anyway

- LLMs are smart, but they can't talk to your systems out of the box

Why do we need this anyway

- MCP helps to integrate existing systems, lots of them have MCP Servers by now
- A lot of AI tools can use MCP Servers
- But what about your own systems or systems without MCP Servers?

- Goal of this session: connect any REST API with (OpenAPI) spec to Copilot, step by step

How does MCP actually work?

- MCP = Model Context Protocol — a standard for giving AI more context, including tools to call
- Tools have a name, description, and input schema → Copilot picks the right one
- In VS Code: tools show up in Copilot chat, Copilot decides when to call them

Using MCP tools in VS Code

- Option A: **Standalone / Remote MCP server** — separate process or remote endpoint, registered via mcp.json
  - Good for language-agnostic setups, shared across editors
  - Not the focus of this session

- Option B: **VS Code extension with LanguageModelTools API** (https://code.visualstudio.com/api/extension-guides/ai/tools)
  - Tools live inside a VS Code extension
  - Tighter integration, easier to distribute via marketplace
  - TypeScript-based, uses the VS Code extension API
  - Our focus today

Using MCP tools in VS Code

days of central knowledge 2026

```
export function registerTools(context: vscode.ExtensionContext, credentialStore: CredentialStore, repositoriesManager: RepositoriesManager) {
    registerFetchingTools(context, credentialStore, repositoriesManager);
    registerSearchTools(context, credentialStore, repositoriesManager);
    context.subscriptions.push(vscode.lm.registerTool(ActivePullRequestTool.toolId, new ActivePullRequestTool(repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(OpenPullRequestTool.toolId, new OpenPullRequestTool(repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(PullRequestStatusChecksTool.toolId, new PullRequestStatusChecksTool(credentialStore, repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(CreatePullRequestTool.toolId, new CreatePullRequestTool(credentialStore, repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(ResolveReviewThreadTool.toolId, new ResolveReviewThreadTool(repositoriesManager)));
}

function registerFetchingTools(context: vscode.ExtensionContext, credentialStore: CredentialStore, repositoriesManager: RepositoriesManager) {
    context.subscriptions.push(vscode.lm.registerTool(FetchIssueTool.toolId, new FetchIssueTool(credentialStore, repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(FetchLabelsTool.toolId, new FetchLabelsTool(credentialStore, repositoriesManager)));
    context.subscriptions.push(vscode.lm.registerTool(FetchNotificationTool.toolId, new FetchNotificationTool(credentialStore, repositoriesManager)));
}

function registerSearchTools(context: vscode.ExtensionContext, credentialStore: CredentialStore, repositoriesManager: RepositoriesManager) {
    context.subscriptions.push(vscode.lm.registerTool(SearchTool.toolId, new SearchTool(credentialStore, repositoriesManager)));
}
```

A quick excurse: OpenAPI

- A standard way to describe REST APIs (endpoints, parameters, responses)
- Why it matters: if your API has a spec, we can auto-generate from it
- Example Tic Tac Toe (https://learn.openapis.org/examples/v3.1/tictactoe.html)

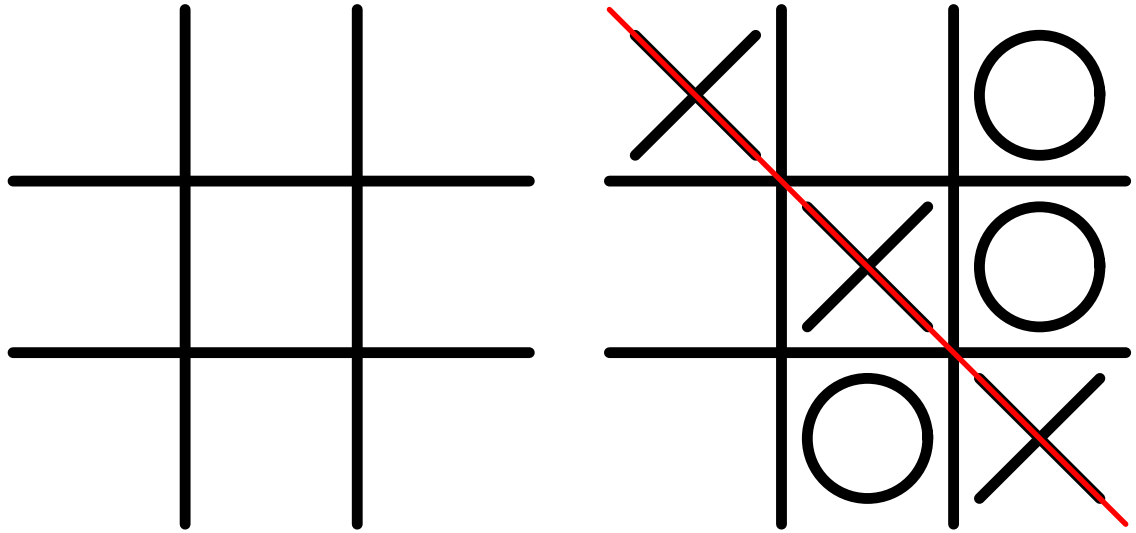


Image source: https://mathworld.wolfram.com/Tic-Tac-Toe.html

A quick excurse: OpenAPI

Metadata

days of central knowledge 2026

```
{
  "openapi": "3.1.0",
  "info": {
    "title": "Tic Tac Toe",
    "description": "This API allows writing down marks on a Tic Tac Toe board\nand requesting the state of
    "version": "1.0.0"
  },
  "tags": [
    {
      "name": "Gameplay"
    }
  ],
```

A quick excurse: OpenAPI

Get data with response types

days of central knowledge 2026

```
"paths": {
  "/board": {
    "get": {
      "summary": "Get the whole board",
      "description": "Retrieves the current state of the board and the winner.",
      "tags": ["Gameplay"],
      "operationId": "get-board",
      "responses": {
        "200": {
          "description": "OK",
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/status"
```

A quick excurse: OpenAPI

Define types

days of central knowledge 2026

```
"winner": {
  "type": "string",
  "enum": [".", "X", "O"],
  "description": "Winner of the game. `.` means nobody has won yet.",
  "example": "."
},
"status": {
  "type": "object",
  "properties": {
    "winner": {
      "$ref": "#/components/schemas/winner"
    },
```

A quick excurse: OpenAPI

Get data with parameters, also error responses


```
"/board/{row}/{column}": {
  "parameters": [
    {
      "$ref": "#/components/parameters/rowParam"
    },
    {
      "$ref": "#/components/parameters/columnParam"
    }
  ],
  "get": {
    "summary": "Get a single board square",
    "description": "Retrieves the requested square.",
    "tags": ["Gameplay"],
    "operationId": "get-square",
```

```
"responses": {
  "200": {
    "description": "OK",
    "content": {
      "application/json": {
        "schema": {
          "$ref": "#/components/schemas/mark"
        }
      }
    }
  },
  "400": {
    "description": "The provided parameters are incorrect",
    "content": {
      "text/html": {
```

A quick excurse: OpenAPI

Parameters are also typed


```
"components": {
  "parameters": {
    "rowParam": {
      "description": "Board row (vertical coordinate)",
      "name": "row",
      "in": "path",
      "required": true,
      "schema": {
        "$ref": "#/components/schemas/coordinate"
      }
    },
    "columnParam": {
      "description": "Board column (horizontal coordinate)",
      "name": "column",
      "in": "path",
```

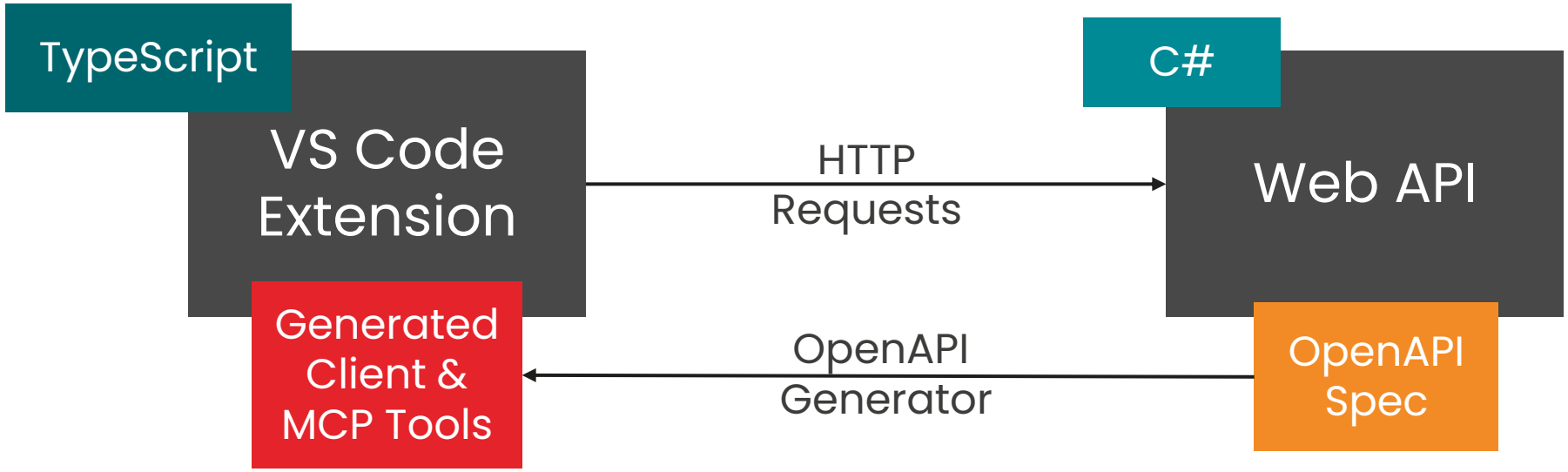
Scope summary

- LLMs need something to talk to your systems, MCPs are often used
- Not every system has an MCP
- But if it has an API, ideally specified via OpenAPI, creating one is easy
- Having it as part of a VS Code extension makes it more powerful

days of central knowledge 2026

Live Demo

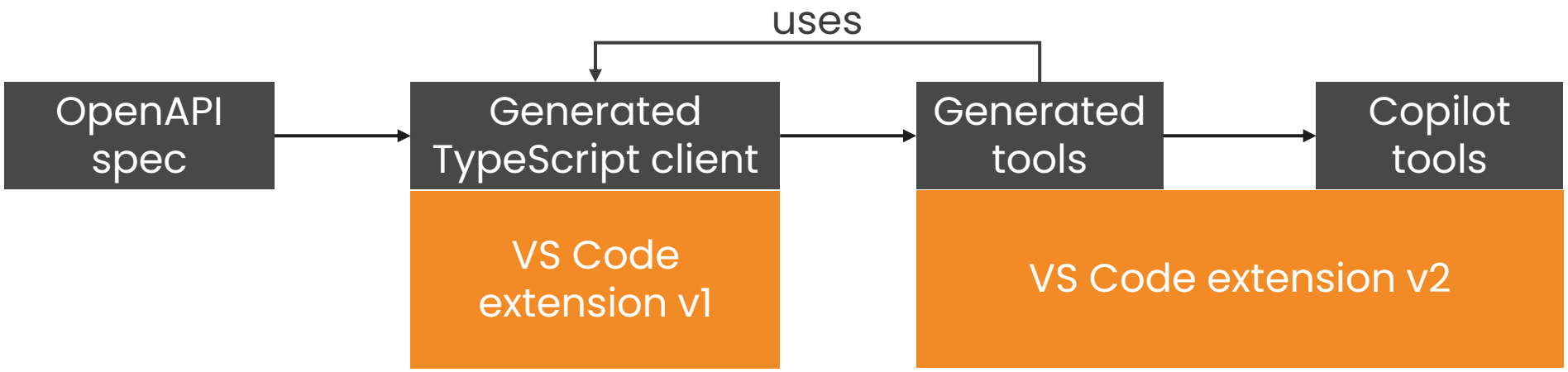
Architecture Overview



days of central knowledge 2026

Live Demo

Architecture Overview



Tips for Real-World Use

- Not every endpoint should be a tool – use a configuration file to in- or exclude
- OpenAPI spec matters
  - Good endpoint descriptions improve automatic tool selection
  - Param descriptions helps Copilot to call tools correctly
  - Documented errors allow unattended troubleshooting by Copilot
  - 3rd party API specs could be enriched with good descriptions in an intermediate step
- Generated clients should handle auth - keep secrets out of the chat
- Make regeneration as easy as possible and run whenever the API changes

days of central knowledge 2026

Real World Example

from COSMO Alpaca

- Existing C# API with OpenAPI spec & existing VS Code extension
- VS Code extension already used openapi-generator to generate commands
- Lot's of actions & features that can be manually invoked
- Many VS Code actions are invoked together regularly and benefit from usage via Copilot
  - Example Flow (manually calling 5 actions): Create Repo ➔ Assign customer ➔ Create App ➔ Create Test App ➔ Create Dev Container
  - Example Prompt: *„Create a new Alpaca PTE repo for customer X with app Y and test app Z and setup a dev environment"*

➔ Quick win: All API endpoints are available to Copilot without much effort, just a few excluded

Limitations

- Generated tools only work in VS Code - MCP servers work anywhere where MCP is supported

- Many extensions with many active tools can cause issues
  - Context window pollution is not a big problem with newest models nowadays
  - Many tools make tool selection for Copilot harder
  - MCP servers often support dynamic tool discovery to overcome this
  - VS Code handles this with virtual tool clustering and pre-selecting likely tools

Wrap Up

days of central knowledge 2026

- Good API docs = good AI tools — invest in your OpenAPI specs and use AI to document them
- Not every API endpoint is a good fit for a tool
- Communication between AI and other systems is key for using AI productively

➔ Try it out with one of your APIs!

https://github.com/lippertmarkus/demo-dok-central-2026

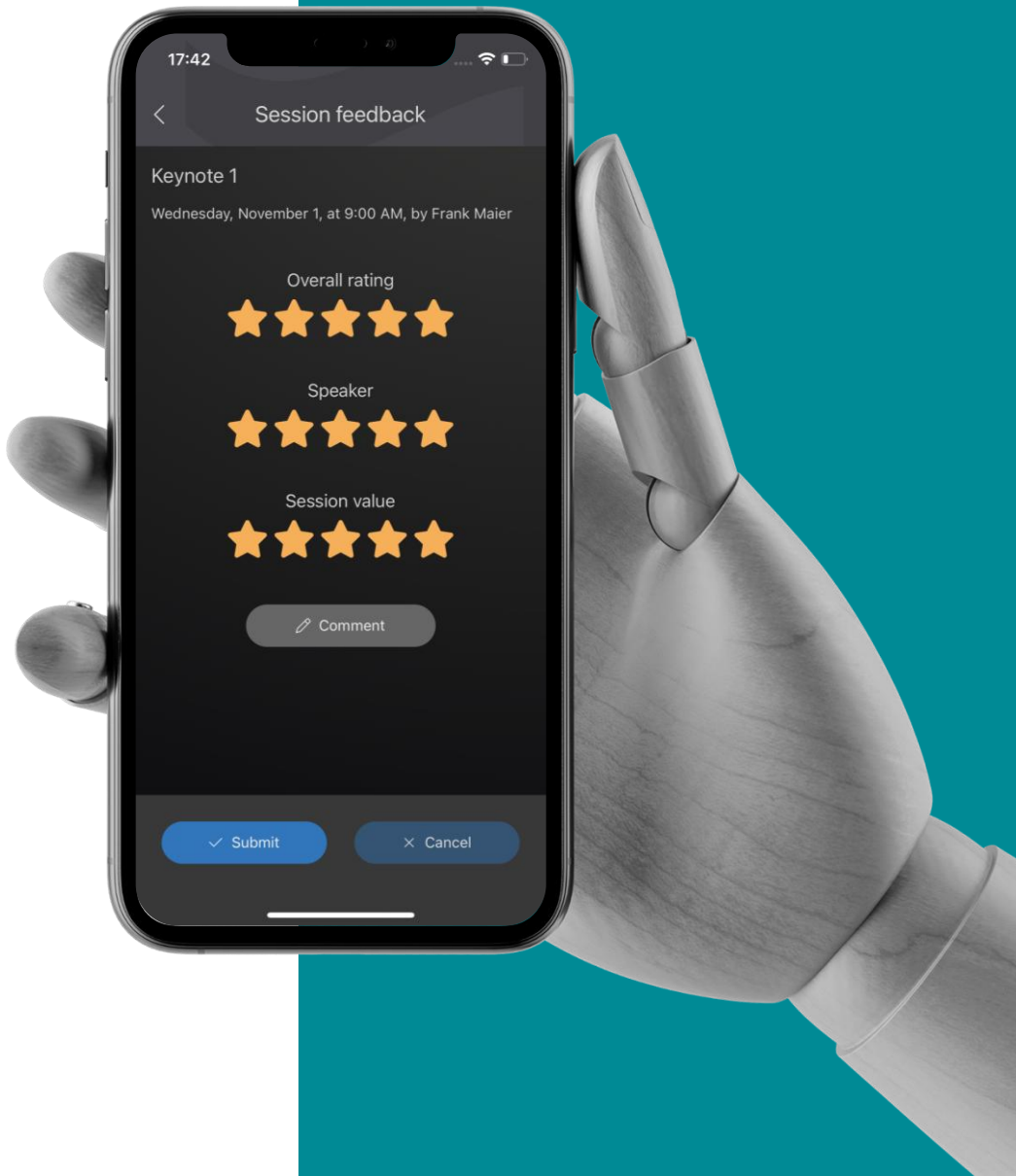
days of central knowledge 2026

Give us Feedback!

and help us improve!

Please rate out session in the Conference App!

and leave a constructive comment.



Thank you !