

Forum: [Hack Forums](#)

Thread: [The HF API MCP server](#)

NoSuchAgency - April 23, 2026, 12:37 pm

HF MCP Is Live:

Hack Forums API Tools For Agents

I have been working on a proper MCP server for the Hack Forums API v2.

It is called hf-mcp, and the repo is here:

https://github.com/CortaLabs/hf_mcp

Current release line: 0.3.1.

The short version: this lets an MCP-capable AI client talk to HF through structured API tools instead of scraping pages, guessing selectors, or trying to drive a browser like a confused intern.

You install the package, authorize your HF account through OAuth, run the MCP server, and then your client can use named tools for HF reads and guarded writes.

The current release also adds a real exploration flow. An agent can now start with zero known IDs, call ``forums.index``, pick a real forum from the packaged catalog, and follow ``_hf_flow.next_actions`` into ``forums.read``, ``threads.read``, and ``posts.read``.

That is the point. Less weird glue code. Fewer random scripts. A cleaner base layer for people who want to build actual HF-aware assistants that can explore, read, draft, and move through HF context without every ID being hand-fed.

What It Can Do Right Now

The read side is already useful.

Current read tools cover the stuff an agent needs to actually understand what is going on:

Account context - read your own HF account state through ``me.read``.

Users - inspect public user data by UID.

Root discovery - call ``forums.index`` / ``forums_index`` when you do not know any IDs yet.

Forums - read forum sections by FID.

Threads - browse threads inside a forum.

Posts - read posts inside a thread, including post bodies when needed.

Contracts - inspect contract data available to your authorized account.

Disputes - inspect dispute records tied to contracts where your token has access.

B-ratings - read rating history where your token has access.

Bytes - inspect Byte activity and balances where supported.

Signature Market (read) - read-side coverage exists in the tool surface, but I am not claiming live-proven success for every Signature Market path yet.

The registered MCP surface is concrete right now. Exposed tools map to actual implemented handlers.

The core browsing path is the important part: an assistant can start from ``forums.index`` without knowing any IDs, then move from forum, to thread, to posts, to users without needing you to manually feed it every selector.

Every flow-aware response carries an additive ``_hf_flow`` object in ``structuredContent``. That object has breadcrumbs and ``next_actions``, so one tool result can tell the agent the valid next tool calls.

Concrete path:

Code `forums.index -> forums.read -> threads.read -> posts.read`

Example zero-ID entry call:

```
Code {
  "tool": "forums.index",
  "arguments": {
    "view": "tree"
  }
}
```

Example flow metadata shape:

Code `structuredContent`:

```
{
  "forums_index": [
    {
      "fid": 375,
      "name": "API",
      "type": "forum"
    }
  ],
  "_hf_flow": {
    "entry_tool": "forums.index",
    "breadcrumbs": ["forum_catalog", "forums.index"],
    "next_actions": [
      {
        "tool": "forums.read",
        "arguments": {
          "fid": 375
        }
      },
      "reason": "Read this forum and discover threads."
    ]
  }
}
```

``forums.index`` is local package catalog data, not a live scrape of the HF forum tree. It is intentionally useful as a root map, and it can drift between package updates if HF changes the forum structure.

Selector handling is also less annoying now. The common HF IDs (``uid``, ``fid``, ``tid``, ``pid``) stay canonical, and the extended tools document the compatibility aliases they actually accept, like ``contract_id``, ``dispute_id`/`did``, and ``oid`` for Signature Market orders.

Read output is designed for both humans and scripts. You do not have to choose between pretty

text and usable data:

Readable by default - tools return a short plain-English summary for the client UI.

Structured for automation - every read still exposes canonical `structuredContent`, so scripts and agents can parse stable JSON instead of scraping text.

Raw when you need it - pass `output\_mode="raw"` or `include\_raw\_payload=true` to attach the exact upstream API payload as an MCP resource.

Those three modes are intentional, not accidental. `readable` is for people watching an agent session, `structured` is for scripts and downstream agents, and `raw` is for audit/debug work when you need the exact upstream payload in view.

Formatting is part of this workflow too. The tooling supports readable output cleanup and Markdown-to-MyCode preparation so draft content can be shaped before it ever touches a live thread.

The flow envelope is not limited to the forum path. `\_hf\_flow` is emitted by `forums.index`, core reads, supported extended reads, local draft/preflight tools, and successful results from implemented guarded writes after confirmed or stubbed execution. Unsupported write families stay absent.

Example read call:

```
Code {
  "tool": "posts.read",
  "arguments": {
    "tid": 6324346,
    "per_page": 1,
    "include_post_body": true,
    "output_mode": "raw",
    "include_raw_payload": true
  }
}
```

That comes back as normal MCP content: a readable text item, top-level `structuredContent`, and an additive raw JSON resource when requested. Post bodies are included when you ask for them. The raw payload is not jammed into the summary text.

Readable mode is the default. It is meant for humans watching the agent work:

```
Code {
  "tool": "threads.read",
  "arguments": {
    "fid": 375,
    "tid": 6324346,
    "output_mode": "readable"
  }
}
```

Example readable output for this launch thread:

Quote:threads.read returned 1 thread(s).

The HF API MCP server

Thread fields

- tid: 6324346
- fid: 375
- subject: The HF API MCP server
- dateline: 1776973048
- uid: 2047020
- username: NoSuchAgency
- views: 88
- lastpost: 1776978491
- sticky: 0

First post fields

- firstpost.pid: 62946370
- firstpost.author.uid: 2047020
- firstpost.author.username: NoSuchAgency

Thread body

HF MCP Is Live:
Hack Forums API Tools For Agents

[
https://raw.githubusercontent.com/CortaLabs/hf_mcp/main/docs/hf_mcp_banner.png](https://cortals.github.io/hf_mcp/)

I have been working on a proper MCP server for the Hack Forums API v2.

It is called hf-mcp, and the repo is here:

[https://github.com/CortaLabs/hf_mcp](https://github.com/CortaLabs/hf_mcp)

The short version: this lets an MCP-capable AI client talk to HF through structured API tools instead of scraping pages, guessing selectors, or trying to drive a browser like a confused intern.

The live tool returns the complete formatted thread body in this section. This example stops here only to avoid recursively embedding the launch post inside itself.

That is the part you want in the chat window while supervising an agent. The same response still carries `structuredContent` behind it, but readable mode should not make the human dig through JSON just to see what happened.

Structured mode is better when you are writing scripts or agent workflows that only need stable fields:

```
Code {  
  "tool": "posts.read",  
  "arguments": {  
    "tid": 6324346,  
    "per_page": 1,  
    "include_post_body": true,  
    "output_mode": "structured"  
  }  
}
```

Example structured output:

Code content:

```
- type: text
  text: "posts.read returned 1 row(s)."
```

structuredContent:

```
{
  "posts": [
    {
      "pid": "62946370",
      "tid": "6324346",
      "fid": "375",
      "subject": "The HF API MCP server",
      "message": "[size=xx-large][align=center][css=68]HF MCP Is Live..."
    }
  ]
}
```

Raw mode is for debugging, auditing, or building tooling that needs the exact upstream HF API payload too:

```
Code {
  "tool": "posts.read",
  "arguments": {
    "tid": 6324346,
    "per_page": 1,
    "include_post_body": true,
    "output_mode": "raw",
    "include_raw_payload": true
  }
}
```

Example raw output:

Code content:

```
- type: text
  text: "posts.read returned 1 row(s)."
```

```
- type: resource
  resource:
    uri: "hf-mcp://raw/posts.read"
    mimeType: "application/json"
    text: "{\"posts\": [{\"pid\": \"62946370\", \"tid\": \"6324346\", \"subject\": \"The HF API MCP server\", \"message\": \"[size=xx-large][align=center][css=68]HF MCP Is Live...\"}]}"
```

structuredContent:

```
{
  "posts": [
    {
      "pid": "62946370",
      "tid": "6324346",
      "subject": "The HF API MCP server",
      "message": "[size=xx-large][align=center][css=68]HF MCP Is Live..."
    }
  ]
}
```

The write side exists too, but it is deliberately gated. Thread creation, post replies, and Bytes

actions require explicit live confirmation before they run.

That is not a limitation I am trying to hide. It is there because this touches a real HF account.

Write coverage that is exposed today is concrete:

Thread create (``threads.create``)

Post reply (``posts.reply``)

Bytes transfer/deposit/withdraw/bump (``bytes.transfer``, ``bytes.deposit``, ``bytes.withdraw``, ``bytes.bump``)

Write coverage that is not exposed right now is also explicit:

``contracts.write`` stays out of the registered surface until sandbox proof exists.

Signature Market write operations are unsupported/unexposed in the public tool registry.

Admin high-risk write surfaces are unsupported/unexposed.

PM body-management tools are unsupported/unexposed.

If you need PM context, use ``me.read`` with Advanced Info. It can expose counters like ``unreadpms`` and ``totalpms``, but it does not expose PM message bodies.

For write flows, I recommend using disk drafts plus preflight checks first, then explicit confirmation for the final live call. That keeps accidental post/transfer mistakes down.

Successful guarded writes also return ``_hf_flow`` when there are meaningful follow-up reads. For example, a confirmed thread create can point back to ``threads.read``, ``posts.read``, and ``forums.read``. The confirmation gate still comes first; flow metadata is not an auto-publish mechanism.

On text rendering: HF may canonicalize double quotes and similar characters into HTML entities on readback. I treat that as normal HF security/rendering behavior, not some bug to bypass.

Drafts And MyCode Prep

This is the part I care about most for actual agent use.

An agent should not be raw-posting whatever it just generated. The better flow is:

Write the reply in Markdown, because that is what most agents produce cleanly.

Run ``formatting.preflight`` to convert it into HF MyCode and save a local draft artifact.

Inspect the draft with ``drafts.read``, or list/review drafts with ``drafts.list``.

Mark the draft metadata with ``drafts.update`` when it is ready or approved.

Publish with ``threads.create`` or ``posts.reply`` by passing ``draft_id`` or ``draft_path``, plus ``confirm_live=true``.

That gives you a real approval point. The draft lives on disk, the MyCode body can be checked before HF ever sees it, and the final write still needs explicit confirmation.

Draft and preflight tools also participate in ``_hf_flow``, but their next actions stay local by default: inspect, update, preflight again, or delete the draft. They do not point an agent straight into live posting.

The Markdown-to-MyCode conversion is practical, not decorative. Basic Markdown headings, bold text, links, quotes, and lists are translated into forum-safe MyCode so public posts look like forum posts instead of pasted agent sludge.

There is no hidden scheduler auto-posting later. Draft metadata can track status and timing notes, but `hf-mcp` does not wake up on its own and mutate your account.

Setup path:

Code `pip install hf-mcp`

`hf-mcp setup init`

`hf-mcp auth bootstrap`

`hf-mcp doctor`

`hf-mcp serve`

The setup flow is:

Create your own Hack Forums API developer app from the HF user control panel.

For the OAuth redirect, you can use the hosted GitHub Pages callback at `https://cortalabs.github.io/hf_mcp/oauth_callback.html`, or you can host the included `docs/oauth_callback.html` yourself.

Put the client ID and client secret in local environment/config only. Do not paste secrets into client prompts or forum posts.

Run `hf-mcp setup init` to create the local config.

Run `hf-mcp auth bootstrap` to authorize your account and save the local token.

Run `hf-mcp doctor` before serving. It checks config path, token path, token presence, and whether the runtime is ready.

Point your MCP client at `hf-mcp serve`.

Example local `.env` values:

Code `HF_MCP_CLIENT_ID=your_app_client_id`

`HF_MCP_CLIENT_SECRET=your_app_client_secret`

`HF_MCP_EXTERNAL_REDIRECT_URI=https://cortalabs.github.io/hf_mcp/oauth_callback.html`

If you host your own callback page, set `HF_MCP_EXTERNAL_REDIRECT_URI` to your own HTTPS URL instead.

Minimal client command:

Code `hf-mcp serve`

If your client wants the command as JSON, the shape is basically:

```
Code {
  "command": "hf-mcp",
  "args": ["serve"]
}
```

Why MCP Works Well For HF

MCP is a clean way to expose tools to AI clients.

Instead of saying "go browse this website" and hoping the agent clicks the right thing, you give it tools with names, inputs, and structured JSON output.

HF maps nicely to that model.

Threads have TIDs. Posts have PIDs. Forums have FIDs. Users have UIDs. The API already has a real object model, so the MCP can expose that directly.

That opens up some interesting uses:

A personal HF assistant that can browse threads and summarize what changed.

A drafting helper that can prepare replies while leaving the final post under your control.

A marketplace helper that can inspect contracts, b-ratings, Bytes context, and user data before you act.

A moderation or research helper that can collect thread context without copy-paste work.

A forum-native agent that stays inside the account scopes you actually authorized.

That last part matters. The tool should not pretend it has more access than your token gives it.

What Ships In The Repo

The repo is meant to be usable by builders, not just something that technically runs on my machine.

It includes:

A Python package published as `hf-mcp`.

A CLI entrypoint: `hf-mcp`.

Setup and auth bootstrap commands.

A local doctor command to verify config, token state, and runtime readiness.

A stdio MCP server for MCP clients.

Docs for configuration, auth, tools, security model, and examples.

Packaged usage skills for agents and clients that support skills.

A maintained forum catalog and `\_hf\_flow.next\_actions` contract so tools can feed the next tool call.

The goal is for someone to be able to install it, connect it, and understand what each tool does without digging through random source files.

There is also a coverage matrix in the docs so the project is honest about what is concrete, what is scope-gated, and what still needs more work.

Use It Responsibly

This is still a live account integration.

Please do not use it for spam, mass posting, harassment, scraping around access controls, or anything that turns HF into a prompt playground.

Keep write tools gated. Check what an agent is about to post or transfer. Use the scopes you actually need, not every scope you can get.

Basically: build cool stuff, but do not be stupid with a real forum account.

Where To Check It Out

GitHub:

https://github.com/CortaLabs/hf_mcp

Package:

Code `pip install hf-mcp`

If you build HF tools, AI assistants, marketplace helpers, or anything that needs clean API access to forum data, this is the base layer I wanted to exist.

It is still early, but the foundation is there now: OAuth bootstrap, zero-ID forum discovery, compounding read flow, safe read tools, guarded write tools, local draft/preflight workflow, docs, and a package structure that should be easy to build on.

If you try it, I would especially like feedback on tool ergonomics. The target is simple: an agent should be able to start from nothing, browse and work with HF, and keep moving from one tool result to the next without fighting the API shape or guessing what a tool returns.

SIM-2470866 - April 23, 2026, 2:01 pm

Very neat! Didn't knew we'd see a HF MCP.

Great work OP, I hope your work gets noticed by Omni

I'm going to try it out and see how it works, specially the summarize thing for threads. That seems like a great addition.

NoSuchAgency - April 23, 2026, 2:08 pm

(Apr 23, 2026 02:01 PM)SIM-2470866 Wrote: [\[CITE\]](#) Very neat! Didn't knew we'd see a HF MCP.

Great work OP, I hope your work gets noticed by Omni

I'm going to try it out and see how it works, specially the summarize thing for threads. That seems like a great addition.

Thanks dog, it's still early in development for sure, but I made it so you can build shit around the MCP.

In fact, if you have Claude subscription, you can use the MCP directly in Claude desktop.

I'd just have the AI help you set it up.

Small live test from the MCP write path.

This is an automated reply using hf-mcp. It was drafted after reading the thread context through the structured MCP tools, then sent through the guarded `posts.reply` helper with explicit live confirmation.

That is the important part for the project: reads work cleanly, and writes still require an intentional `confirm\_live=true` step before touching the forum.

0.2 update is live.

I pushed the 0.2 release line with a cleaner read-output contract for MCP clients.

The big change: read tools now default to a human-readable summary while still returning canonical structured data for agents and scripts. So a client can show something readable like:

Code posts.read returned 1 row(s).

while still receiving protocol-level structured output like:

```
Code {
  &quot;posts&quot;: [
    {
      &quot;pid&quot;: &quot;62946370&quot;,
      &quot;tid&quot;: &quot;6324346&quot;,
      &quot;subject&quot;: &quot;The HF API MCP server&quot;,
      &quot;message&quot;: &quot;...post body when requested...&quot;
    }
  ]
}
```

```
]
}
```

For automation, you can still call tools with ``output_mode="structured"``. For debugging or exact API parity, use ``output_mode="raw"`` or ``include_raw_payload=true`` to attach the upstream HF API response as an MCP resource.

I also updated the docs/thread copy with concrete examples for readable, structured, and raw output modes, plus clearer setup notes for creating your own HF API app and using either the hosted GitHub callback or your own hosted callback page.

Posted via hf-mcp.
Codex Agent making the post for me [\(Click to View\)](#)? loading encrypted image...
HF MCP formatting probe.

Being transparent: the last update exposed a formatting issue where double quotes in JSON examples can show up as ``"`` after the HF write/render/readback path.

This post is intentionally testing that behavior through the MCP write path.

Double-quote JSON example:

```
Code {
  &quot;tool&quot;: &quot;posts.read&quot;,
  &quot;arguments&quot;: {
    &quot;tid&quot;: 6324346,
    &quot;output_mode&quot;: &quot;structured&quot;,
    &quot;include_post_body&quot;: true
  }
}
Inline double-quote check: `output_mode=&quot;structured&quot;` and
`output_mode=&quot;raw&quot;`.
```

Quote-light alternative that may be safer for HF posts:

```
Code structuredContent:
  posts:
    - pid: '62946370'
      tid: '6324346'
      output_mode: 'structured'
      include_post_body: true
Inline single-quote check: output_mode='structured' and output_mode='raw'.
```

If the first example shows entities but the second stays readable, that points to HF/MyCode quote handling rather than the MCP JSON transport itself. Posted via hf-mcp as a live formatting probe.
Tiny HF MCP quote-entity probe.

Testing one possible escape hatch for code examples posted through the HF API write path.

Raw double-quote control:

```
Code {
  &quot;tool&quot;: &quot;posts.read&quot;
}
```

Decimal numeric entity version:

```
Code {  
  &quot;tool&quot;: &quot;posts.read&quot;;  
  &quot;output_mode&quot;: &quot;structured&quot;;  
}
```

Inline numeric entity check: `output_mode="structured";`

Goal: see whether `"` renders/copies as a real double quote on the HF page, even if the API source/readback stores an entity. Posted via hf-mcp as a live formatting probe.

Follow-up on code snippets posted through the API.

We chased the double-quote formatting issue a bit deeper instead of hand-waving it away.

Short version: this looks like a natural Hack Forums API safety/sanitization boundary for API-written post content, not the MCP JSON transport randomly mangling text.

What we verified:

The MCP write helper now normalizes host-provided HTML entities before transport.

The transport sends normal JSON payloads; it is not introducing the entity text.

Stock MyBB code-block parsing does not appear to be the source of this exact behavior.

Live HF API write/read probes showed raw double quotes in code examples coming back as `"`.

Trying decimal numeric quote entities did not work as an escape hatch; HF canonicalized those back to `"`, too.

So for now, I am treating quote conversion in API-written code examples as expected defensive handling on the HF side. That is honestly reasonable for a forum API: live post writes should be conservative about text that can become markup or rendered page content.

Practical impact:

For public update posts, I will avoid quote-heavy JSON examples when they need to be copy-paste exact.

For examples that must be exact code, better options are paste links, attachments, screenshots, or quote-light formats.

For MCP/tool consumers, the client can still decode returned entities for readable output while preserving the raw HF API payload for debugging.

I also documented this in the packaged hf-mcp skills so downstream users and agents do not rediscover it the hard way. The repo now includes guidance under the public skills directory for MyCode, API-v2 behavior, and guarded write usage.

Posted via hf-mcp.

hf-mcp formatting and readability update

I cleaned up another important part of hf-mcp: how forum content is read by agents, how agent-written content is prepared for Hack Forums, and how the MCP handles HF's normal escaping behavior.

Why this matters

Agents are much better at working with clean Markdown than raw MyCode. Hack Forums is still a MyCode forum. This update makes hf-mcp bridge those two worlds instead of forcing every client to

solve formatting on its own.

Readable read output

Read tools now support `body_format` for post-style content.

`body_format markdown` gives agents a readable Markdown version of post bodies.

`body_format clean` strips formatting noise and returns plain text.

`body_format raw` preserves the original HF MyCode for exact inspection or debugging.

The default is meant to be useful for desktop chat clients: readable enough for the agent to reason over, while structured output is still available for automation.

Markdown write input

Write tools now support `message_format`.

`message_format mycode` remains the default, so existing callers keep working.

`message_format markdown` lets an agent draft naturally in Markdown.

hf-mcp converts common Markdown into HF MyCode before sending the write.

That means bold text, lists, links, quotes, images, and code-style content can be drafted in the format most agents already use, while the forum still receives normal MyCode.

This post is using the new path

This reply was written as Markdown and sent through `posts.reply` with `message_format markdown`. The section headings, list formatting, and project link below are being converted by hf-mcp before the live write reaches HF.

Project repo: https://github.com/CortaLabs/hf_mcp

Quote and entity handling

We also investigated the double-quote behavior from API-written examples. HF naturally escapes some characters in stored or returned post content. That is expected for a forum-side safety boundary.

hf-mcp now handles the common read-side case more cleanly:

Double-quote entities are decoded for agent-facing body output.

Raw output remains available when you need to inspect what HF actually returned.

The fix is intentionally narrow, so the read path does not become a broad or surprising HTML decoder.

I also noticed arrow entities in raw MyCode. The Markdown read path presents those normally for agents, while raw mode still preserves what came back from HF.

Packaged skills and downstream guidance

The package now includes updated skills for downstream users and MCP-capable agents.

hf-mcp-reads explains the `body_format` options.

hf-mcp-writes explains the `message_format` options.

hf-mycode covers practical MyCode authoring guidance.

That should make the package easier to use from desktop chat clients without needing someone to

already know all the local project decisions.

Version summary

0.2.4 added the read and write formatting layer.

0.2.5 added narrow quote-entity normalization for body output.

0.2.6 fixed inline Markdown conversion so names with underscores are preserved correctly during Markdown-to-MyCode conversion.

Write safety direction

The next improvement is local draft validation before live writes. A live forum should not be the first place malformed generated MyCode gets tested.

The direction I want is:

Prepare the draft locally.

Preview the final MyCode before it touches HF.

Let the user approve the exact content.

Publish only after approval.

That keeps agents useful for drafting and formatting while keeping the human firmly in control of what gets posted.

hf-mcp 0.2.8 update: draft-first forum posting is now in.

We just finished a pretty important quality pass on the MCP write workflow. The big change is that agents no longer need to jump straight from generated text to a live Hack Forums post.

The new flow is draft-first:

Write the post naturally in Markdown.

Run a local formatting preflight.

Let hf-mcp convert the content into Hack Forums MyCode.

Save a local draft artifact with the converted body, validation report, integrity score, and metadata.

Review or update that draft before anything is posted live.

Publish from a draft id or draft path only when live confirmation is explicitly set.

That gives us a much cleaner approval path. An agent can prepare a post, inspect it, categorize it, mark it ready or approved, and then use the same cached draft for the real write. The live write tools still keep the hard safety gate: no explicit live confirmation means no forum mutation.

What shipped in this pass:

Local formatting preflight for Markdown and MyCode preparation.

Draft listing so cached drafts can be browsed.

Draft reading so a prepared post can be inspected before posting.

Draft metadata updates for title, category, status, and scheduled_at.

Draft deletion with explicit delete confirmation.

Existing post and thread write tools can publish from a draft id or draft path.

Draft-backed writes still run validation before transport.

Mixed direct message plus draft inputs are rejected before anything reaches HF.

Draft publishing does not mark a draft as published or mutate its metadata behind your back.

A few important design choices:

scheduled_at is metadata only. There is no scheduler in this release.

The draft cache is local by default under the hf-mcp config area, and it can be overridden by config or environment.

The feature is cross-platform in the package design: normal Python, virtualenv or conda, native Windows Python, and Windows desktop clients bridging into WSL are all documented.

The docs now include first-class Claude Desktop style setup guidance, including the WSL command pattern, while keeping paths and distro names user-specific.

We also documented the Hack Forums quote/entity behavior from the live probes. HF appears to canonicalize double quotes in API-written content as part of its normal security and sanitization boundary. That is not something hf-mcp should try to bypass. The practical answer is to avoid quote-heavy public code snippets when exact copy and paste fidelity matters, or use safer delivery formats such as quote-light examples, paste links, attachments, screenshots, or read-side decoded representations for agents.

This update is pushed as 0.2.8, and the package build now includes the docs plus all shipped hf-mcp skills in both the source distribution and wheel.

This post itself was prepared through the new draft and preflight path before live posting, which is exactly the workflow this release is meant to support.