

If you spend any time working with entity resolution, factual lookup, or record linking inside an MCP client, you learn quickly that not every failure is a search failure. Sometimes the model asked a vague question. Sometimes the record really is ambiguous. Sometimes the data source is available, but the optional cross-check is not. And sometimes the system is healthy, but operating under a narrower configuration than you assumed.

That is why `kg_status` matters in the Wikidata + Google Knowledge Graph MCP server.

This project, published as an open-source MCP server and CLI, sits in a practical space that many teams care about: searching Wikidata, reading selected facts, and linking local records to Wikidata QIDs with inspectable evidence and explicit uncertainty when the evidence is not strong enough. It works with MCP clients such as Claude Code, Cursor, and Codex. It is read-only. It does not edit Wikidata, Google, or user data. It is not official Wikimedia or Google software. Those boundaries are not minor details. They shape how every tool in the server should be interpreted, including `kg_status`.

A lot of people look first at `kg_search` or `kg_resolve`, because they are the visible, action-oriented tools. `kg_status` tends to be treated as plumbing. In practice, it is closer to an operating panel. It tells you what environment you are actually querying, which matters more than most users expect.

Why a status tool matters in this MCP server

The design of this server is unusually deliberate about scope and evidence. It does not dump huge result sets into the client. It uses bounded search, returning 3 candidates by default and at most 5. It supports selected-fact retrieval, including ranks, qualifiers, and references on request. Its resolution logic is deterministic, with explicit outcomes such as `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. It can optionally cross-check against the Google Knowledge Graph Search API using exact identifier joins, specifically `/m/` matched through Wikidata property `P646` and `/g/` matched through `P2671`. At the same time, the project is careful not to overclaim. Agreement between Google and Wikidata is treated as provider concordance, not proof of identity.

Those are strong design choices, but they create a practical question for anyone using the server inside an agent workflow: what is available right now, and under what conditions?

That is the gap `kg_status` helps fill.

In systems like this, a status endpoint or status tool is rarely just a heartbeat. It is the fastest way to answer operational questions before you start interpreting the output of search and resolution tools. If you skip that step, you can easily misread results. A weak result might reflect ambiguous source data, or it might reflect the current backend setup. Those are very different situations.

What `kg_status` tells you, even before you inspect its exact payload

The verified documentation confirms that `kg_status` is one of the documented MCP tools. It also confirms several facts about the system around it: Wikidata requires no account or API key, the Google Knowledge Graph Search API is optional, and the server can be used in different MCP clients.

From that alone, there are some solid, defensible things `kg_status` is there to clarify.

First, it helps establish whether you are operating in a Wikidata-only mode or in a configuration where the optional Google cross-check is available. That matters because a user can ask for “MCP for google knowledge graph and wikidata,” but the running instance may only have the Wikidata side active. Without a status check, you might assume the absence of Google concordance means the entity is weakly grounded, when the simpler explanation is that the Google side is not configured.

Second, it helps set expectations for how to interpret downstream tool behavior. This project does not claim to be an export of the Google Knowledge Graph. It uses the Google Knowledge Graph Search API optionally, and it keeps the core identity work anchored in bounded search, selected evidence, and deterministic outcomes. A status tool therefore frames what kind of evidence chain is available in the current session.

Third, it separates environment facts from data facts. That is a subtle distinction, but a crucial one. `kg_search`, `kg_entity`, and `kg_resolve` tell you things about entities, candidates, and evidence. `kg_status` tells you things about the server context in which those answers are being produced.

In practical use, that separation keeps teams from blaming the data when the real issue is configuration.

The hidden operational question behind every match result

When a resolver returns `AUTO_MATCH`, most users focus on the confidence implied by the label. When it returns `HOLD` or `AMBIGUOUS`, they focus on uncertainty. But experienced users ask **Learn here** a prior question: what evidence channels were actually in play?

That is where `kg_status` becomes more than a convenience.

The server’s resolution logic is deterministic, which is a strong trait for production work. Deterministic behavior makes results easier to test, easier to explain, and easier to audit. But determinism is only fully meaningful if you know the state of the system that produced the result. If your workflow depends on optional Google concordance and that concordance is not available in the current setup, the deterministic result is still valid, but it is valid within a narrower evidence context.

I have seen versions of this mistake in many data pipelines, even outside MCP. Someone runs a linker in development with one set of connections, moves it into a shared environment with another set, and then spends hours comparing “quality” across runs when the deeper cause is simple capability drift. A status tool is the short path around that mess.

With MCP for google knowledge graph and wikidata, the environment can be straightforward, but it is not trivial. Wikidata is available without an account or API key. Google is optional. The server supports several tools with distinct purposes. That is exactly the kind of setup where a status check prevents false assumptions.

Why `kg_status` matters more in this project than in generic search wrappers

A lot of search tools expose status in a vague way, often little more than “service up” or “service down.” This project is different because the difference between one backend and two backends changes how a careful user should interpret evidence.

The project’s own design makes this plain. It can perform an optional Google cross-check using exact identifier joins. That is not fuzzy name matching across providers. It is specific concordance work involving `/m/` and `/g/` identifiers tied to Wikidata properties. At the same time, the project explicitly says that

agreement between Wikidata and Google is not proof of identity. That is a mature stance. It recognizes concordance as useful corroboration, not as a magic stamp.

A status tool in that environment is not decorative. It tells you whether that corroboration path is even on the table.

For users exploring MCP for wikidata alone, the answer may be simple: yes, the Wikidata side is active, no special keying required. For users expecting MCP for google knowledge graph as part of the same workflow, status becomes more consequential. It helps prevent a very common mistake, which is reading absence of cross-provider support as absence of cross-provider agreement.

Those are not the same thing.

What you should infer, and what you should not

Good operational judgment depends on respecting the boundary between what the documentation confirms and what you wish it confirmed.

You can safely infer that `kg_status` exists to report the current state of the MCP server in some useful way. You can safely infer that one practical dimension of status is whether the optional Google Knowledge Graph Search API path is available, because the project clearly distinguishes required and optional components. You can safely infer that this matters to search, entity retrieval, and resolution workflows.

What you should not do is invent exact fields, exact response shapes, or exact health semantics that the verified context does not provide. The project documentation available here confirms the tool's existence, not its full schema.

That distinction may sound fussy, but it is exactly the kind of discipline this server encourages. The whole project leans toward inspectable evidence and explicit uncertainty. Treating `kg_status` carefully is part of using the system in the spirit it was built.

How `kg_status` changes the way you read the other MCP tools

The documented tool set includes `kg_search`, `kg_entity`, `kg_related`, `kg_resolve`, and `kg_status`. Look at them together and a pattern appears. This is not just a collection of query endpoints. It is a compact workflow for discovery, inspection, relationship exploration, resolution, and environment awareness.

That last part is easy to overlook.

If you use `kg_search`, the bounded result behavior matters. The server defaults to 3 candidates and caps at 5. That is a thoughtful choice because it keeps the output narrow enough for an agent or analyst to inspect. But bounded search also means you ***Wikidata MCP*** should resist reading a short candidate list as a universal statement about the world. Sometimes it reflects the search design. Sometimes it reflects the current query. Sometimes it reflects what backends are active. `kg_status` helps you know which interpretation is plausible.

If you use `kg_entity`, selected-fact retrieval can include ranks, qualifiers, and references on request. That tells you the system is built for targeted inspection rather than bulk extraction. A status check helps establish that the retrieval environment is what you think it is before you start comparing evidence traces.

If you use `kg_resolve`, the value is even more obvious. A deterministic resolver with explicit outcomes is exactly the sort of tool that benefits from a clear picture of current capabilities. If a record lands in HOLD or

AMBIGUOUS, the next question is not always “what is wrong with the record?” Sometimes it is “which evidence channels were available when we resolved it?”

That is the kind of question `kg_status` supports.

A sensible order of operations

In real-world use, there is a practical rhythm that reduces confusion and makes the server easier to trust.

1. Check `kg_status` to understand the current environment.
2. Use `kg_search` to inspect a bounded candidate set.
3. Use `kg_entity` when you need selected facts, ranks, qualifiers, or references.
4. Use `kg_resolve` when you need a deterministic match outcome.
5. Revisit status assumptions if a result surprises you.

That sequence is not a hard rule, but it reflects how experienced users avoid category errors. They do not jump straight from a surprising match result to a narrative about data quality. They first verify what system state they were actually operating in.

The special role of `kg_status` when Google is optional

The optional nature of the Google Knowledge Graph Search API is the single biggest reason `kg_status` deserves attention.

In many integrations, optional components are where misunderstandings begin. A colleague assumes they are enabled because they were enabled last week. A local environment has a key, a shared one does not. Someone demos a feature in one client and expects identical behavior in another. Nothing is technically broken, but expectations drift.

This project’s architecture makes that drift consequential because optional Google support is not just a speed booster or a nice-to-have. It changes whether provider concordance is available as part of your inspection and resolution story.

The project is also careful to keep that concordance in its proper place. Google and Wikidata agreement is not proof of identity. That is an important guardrail. Still, there is a big difference between “the two providers agree on this identifier mapping” and “we did not consult the second provider because it was not available.” A status check helps you tell those scenarios apart.

That is especially relevant for teams evaluating MCP for google knowledge graph and wikidata as part of a repeatable workflow. If you want consistent interpretation across runs, you need to know whether you are comparing like with like.

Why this matters for auditability and human review

One of the strongest signals in the project description is its emphasis on inspectable evidence and explicit uncertainty. Those are not marketing phrases. They are design choices that support human review.

When a system says “I can’t tell” in a structured way, as with `HOLD`, `AMBIGUOUS`, or `NO_CANDIDATE`, it becomes much easier to decide what a person should inspect next. But good review also depends on context. A reviewer needs to know whether the result came from a Wikidata-only path or a path that included optional Google concordance checks.

That is why `kg_status` belongs in any serious audit trail built around this server.

The CLI support for batch work and evidence export reinforces that point. Once you start exporting evidence or processing records in volume, environment awareness stops being optional. It becomes part of the interpretation layer. If a batch from Tuesday used one capability set and a batch from Thursday used another, a status snapshot becomes the difference between a trustworthy comparison and a misleading one.

`kg_status` is not a shortcut to truth

There is another reason experienced users appreciate status tools: they help constrain ambition.

This server does not claim to solve identity by brute force. It does not claim that Google and Wikidata agreement proves identity. It does not claim to be official software from either ecosystem. It does not edit source systems. It is read-only, bounded, and explicit about uncertainty.

`kg_status` fits that philosophy. It does not tell you which entity is correct. It does not replace inspection of qualifiers, references, or ranks. It does not make ambiguous records unambiguous. What it does is narrower and more valuable: it tells you the operating context within which all of that evidence should be read.

That is often the difference between a robust workflow and a brittle one.

I would go further and say that status awareness is part of epistemic hygiene when you work with linked data tools. If a system gives you structured outcomes and bounded candidate sets, the least you can do is confirm the conditions under which those outputs were generated.

Reading `kg_status` with the right mental model

The best mental model is not “status equals uptime.” It is “status equals current evidence environment.”

That framing aligns with what the server is built to do. It searches Wikidata. It can retrieve selected facts, including metadata like ranks, qualifiers, and references. It can resolve records to Wikidata QIDs with deterministic outcomes. It can optionally cross-check against Google using exact identifier joins. It works in MCP clients where agents may otherwise glide over infrastructure details.

Seen that way, `kg_status` is a guard against silent assumption. It tells you what kind of run you are having before you interpret what the run produced.

For teams adopting MCP for wikidata or a combined MCP for google knowledge graph and wikidata workflow, that small bit of discipline pays off quickly. You get cleaner debugging, more honest comparisons, better review notes, and fewer arguments about whether a surprising result came from the data or from the environment.

That is a lot of value from a tool people often ignore.

The practical takeaway

If you use this server casually, `kg_status` saves you from confusion. If you use it professionally, it saves you from false confidence.

The distinction matters. A casual user may only need to know whether the system is ready enough to search. A professional user needs to know whether the current run supports the evidence model they think

they are using. In a project where Wikidata is always central, Google is optional, search is bounded, fact retrieval is selective, and resolution outcomes are explicit, that context is not a side note. It is part of the meaning of the result.

So when you see `kg_status` in the tool list, do not treat it as filler beside the more glamorous commands. It is the tool that tells you how to read the rest.