Checking Indexing Status Using Google Sheets and App Script

You can build a lightweight, automated robot that checks whether dozens of URLs are in Google's index—all inside a Google Sheet, without opening a browser tab for each page. **Checking Indexing Status Using Google Sheets and App Script** means wiring up the Search Console URL Inspection API through a bit of JavaScript. The setup isn't a pretty SaaS dashboard, but it puts raw inspection data right into cells you can filter, sort, and share. That matters when a client sends 300 "priority" URLs at 5 p.m. on a Friday.

A typical agency sees roughly 20–40% of freshly published article URLs remain unindexed after the first crawl cycle, according to sampling across mid-size content sites monitored via Search Console. Hours of manual clicking turn into five minutes of runtime. The trick is handling the API's 200 responses that say "indexStatusResult":{"verdict":"PASS"} versus the ones that quietly return "verdict":"NEUTRAL"—meaning Google hasn't made a final call yet.

This approach uses nothing beyond a regular Google account with Search Console property access, the built-in Apps Script editor, and a modest daily quota (2,000 queries per property per day for the URL Inspection API, based on Google's published limits). You can test a small batch immediately and scale to hundreds of URLs with batching and incremental checks.

What You're Really Measuring with Index Checks

An "indexed" stamp doesn't mean the page will rank. It means Google has fetched it, parsed it, and decided it's worth storing in the vast, chaotic haystack of the index. The URL Inspection API returns the most recent crawl snapshot: last crawl time, user-declared canonical, Google-selected canonical, mobile usability, and any noindex or coverage errors. That's the signal you're digging for.

Think of it like checking a parcel tracking number. A "DELIVERED" status doesn't tell you if the box ended up under the porch or in the pond, just that the system recorded a terminal state. Similarly, a verdict of "PASS" with a crawl timestamp from three weeks ago could hide the fact that the page's content has since changed and the new version hasn't been reprocessed.

In practice, when you’re auditing a site migration, you’ll see a cluster of old URLs still returning "PASS" while the redirect chain is correctly set. That’s because the API reports the last known state; it doesn’t force a live recrawl. The trick is pairing the inspection result with a live fetch from the Indexing API (if submitting) or simply noting the crawl date and flagging anything older than a threshold you set—say, 14 days.

> Rule of thumb: never treat a single inspection verdict as ground truth for freshness. Cross-reference with the lastCrawlTime field and act accordingly.

Setting Up the Google Sheets Workbook and Script Editor

Create a fresh Sheet. Name the first column A “URL”, column B “Verdict”, C “Last Crawl”, D “User Canonical”, E “Google Canonical”, F “Coverage Issues”, G “Mobile Usability”. That’s your dashboard. Then open **Extensions → Apps Script**.

Inside the script editor, rename the project something like *IndexCheckInspector*. Then copy the Search Console API service name: https://www.googleapis.com/auth/webmasters.readonly. You’ll need to add it under **Project Properties → Scopes** or simply call it during the authorization flow. In the new script, the first function will be a menu builder to attach a custom menu item, say “Check Indexing Status”, to the Sheet’s UI.

One detail that trips up first-timers: the script must be bound to the Sheet, not standalone. Otherwise, SpreadsheetApp.getActiveSpreadsheet() returns null. Also, you’ll need to enable the Search Console API from **Resources → Advanced Google Services** and then switch it on for the project. That step is easy to skip, and the error message—“API not enabled”—is cryptic early on.

Writing the Apps Script Code to Call Search Console API

The core inspection call uses UrlInspection.Index.inspect(). It expects a single URL and an inspection URL (the property’s Search Console address, like https://example.com/). The response is JSON. Below is the minimal working function you’d embed inside a loop that walks down column A.

```javascript function inspectURL(url, siteUrl) { const requestBody = {
```

"inspectionUrl": url, "siteUrl": siteUrl }; // UrlInspection is the key advanced service const response = UrlInspection.Index.inspect(requestBody); return response.inspectionResult; } ```

That returns an object. The client doesn't want raw JSON. So you flatten the important bits and paste them into row cells:

```javascript function processRow(rowData) { const url = rowData[0]; const site = 'https://yoursite.com/'; // pull from a settings cell better const result = inspectURL(url, site); const verdict = result.indexStatusResult.verdict; const lastCrawl = result.indexStatusResult.lastCrawlTime || 'null'; const userCanonical = result.indexStatusResult.userCanonical || ''; const googleCanonical = result.indexStatusResult.googleCanonical || ''; const coverageState = result.indexStatusResult.coverageState || ''; const issues = result.mobileUsabilityResult?.issues?.join(', ') || ''; return [verdict, lastCrawl, userCanonical, googleCanonical, coverageState, issues]; } ```

A common failure: when a URL is completely unexamined (never crawled), the verdict might be "NEUTRAL" and the lastCrawlTime absent, which throws a null pointer if you don't guard. That's why each property needs a safe getter. Also, the API returns a quota-limit error when you exceed 2,000 per day, but it gives you a 429 status inside the Apps Script exception—you need to catch and back off.

The menu setup code looks like this:

```javascript function onOpen() { const ui = SpreadsheetApp.getUi(); ui.createMenu('Indexing Tools') .addItem('Check Selected URLs', 'checkSelectedUrls') .addToUi(); } ```

In checkSelectedUrls(), you'd grab the active range, iterate only over rows that have a URL, and write results back. A full script with error handling, progress reporting, and quota sleep runs around 80 lines.

Common Pitfalls When Working with Quota and Authentication

Authentication fails silently if the Google Cloud Project linked to the script doesn't have the Search Console API enabled. Even if it's enabled in the Apps Script

“Advanced Google Services” panel, the underlying Cloud Project might be the default “My Project” with nothing turned on. Fix: go to console.cloud.google.com, find the project ID that Apps Script shows, enable the Search Console API there, then add the webmasters.readonly scope explicitly.

Quota planning is brutal if you batch 500 URLs and hit the limit on the 347th request. The script will crash with a red banner. The solution is to implement a Utilities.sleep() after every 180–200 calls and, critically, to check the error.message for “Quota” and either stop gracefully or save state in a cell (e.g., “QUOTA_EXCEEDED_ROW_347”). I’ve seen scripts that just overwrite results with blank because the error handler wasn’t placed correctly.

Another real-world snag: the API sometimes returns a verdict of “BLOCKED” when the robots.txt blocks Googlebot. You can spot it instantly, but the property owner might not have alerted you because the robots.txt is fine for most pages—just not the subset you’re auditing. So your sheet flags 20 URLs as “BLOCKED”, and you’ve found a staging server’s robots.txt rule that was accidentally deployed to production.

:::warning Running the script from a consumer Gmail account with multiple Search Console property scopes can cause auth pop-ups if the token stored in the script doesn’t cover all sites. Grant access to all necessary properties in one session, or the script may return a 403 "user does not have sufficient permission" for half your list. :::

Interpreting Inspection Results and Handling Edge Cases

Here’s a quick field guide to outcomes you’ll see. The verdict “PASS” means the URL is indexed, but check coverageState. If it says “Indexed, not submitted in sitemap,” that’s fine unless you rely on sitemap discovery. A verdict of “NEUTRAL” with a recent crawl time might mean Google is evaluating the page; wait a day and recheck. A verdict of “FAIL” almost always points to a crawl anomaly—noindex tag, redirect loop, server error, or blocked by robots.txt.

Mobile usability is a secondary but useful signal. If the API returns “MOBILE_FRIENDLY” with no issues, that’s good. If it’s “NOT_MOBILE_FRIENDLY” with a laundry list of “Viewport not set”, “Content wider than screen”, it’s a direct hint that even though the page is indexed, its chances of performing well out of mobile SERPs are diminished. And over 60% of searches are mobile, according to Google’s own statements.

Edge case: you might inspect a URL and get an empty result object. This occurs if the property URL in the request is incorrect (e.g., you used https://www.example.com but the Search Console property is domain property example.com). The API call won't crash; it will return a null inspectionResult. That's why we always validate that the returned object is not null before indexing its properties.

- **Match the exact property URL:** Use the string as it appears in Search Console.
- **Guard against 429s:** Implement exponential backoff and a safety row marker.
- **Differentiate "NEUTRAL" vs. "PASS" vs. "FAIL":** Create conditional formatting in the Sheet.
- **Store the timestamp:** Hard-code the date of check in a cell so you know when the data is stale.
- **Handle +200 URLs:** Use SpreadsheetApp.flush() periodically to write partial results, preventing a crash from losing all data.

Real-World Scenarios Where This Script Cuts Hours

Before: an SEO manager manually opened the URL Inspection tool for 180 URLs, pasting each, copying the result into a spreadsheet. It took three hours and led to finger fatigue and a missed "BLOCKED" verdict on a key landing page. After: the same sheet, with the script, returned all results in 8 minutes, highlighted two critical blocks, and let the manager alert the developer before the weekend.

Before: a content team launched 50 new blog posts and waited a week, then randomly sampled 10 in Search Console. They assumed all was well. After running the script on the full set, they discovered 12 were "NEUTRAL" with no crawl at all, and 5 had 404 due to slug typos. The team could then submit the correct URLs to the Indexing API and fix the broken links immediately—saving potential ranking decay.

Frequently Asked Questions About This Method

Can I check competitor URLs with this?
No, the Search Console API requires ownership or user permission for the property. It won't return data for sites you don't control.

Is there a free bulk URL index checker alternative?
Yes, some third-party services like SpeedyIndex offer batch checking APIs that

don’t require Search Console access. They usually query Google’s cache or use other indicators; accuracy differs and they aren’t as definitive as the official inspection data.

Why does the script sometimes return “NEUTRAL” even for old pages?
Google may have crawled the page but hasn’t made a final indexing decision, often due to quality signals or crawl budget allocation. It’s a limbo state.

Can I automate export of all my site’s indexed pages?
Not easily via the Inspection API alone. You would be better off using Search Console’s Index Coverage report export, which shows indexed status in aggregate. This script is for specific URL lists.

Do I need a billing account in Google Cloud?
The URL Inspection API has no per-call charge beyond the shared quota, but you must link your Apps Script project to a Google Cloud Project that has the API enabled. No billing is required unless you go above the free tier for other services.

Deciding What to Do After the Check

A “PASS” with recent crawl? You’re done for that URL. A “NEUTRAL” with no crawl? The page likely needs a stronger internal link or a ping. Use the Indexing API to request a crawl. Or push the URL through a fast indexing service if speed matters—trading convenience for certainty.

A “FAIL” with “URL is not on Google” or “Crawling issues” means you’ve got a technical barrier. Open the live URL Inspection in Search Console for that one page to get the full diagnostic, because the API response omits the step-by-step checks you see in the web interface. Keep the script as the triage tool, not the final diagnosis.

If this becomes part of a weekly workflow, store results in a new sheet tab named with the date, so you can track indexation trends. Six weeks of data on the same 200 URLs will reveal patterns: seasonal re-crawls, content decay, or growing coverage issues that the single snapshot hides.

Cited Sources

1. Bing Webmaster. "Submit Sitemaps." bing.com/webmasters
2. Google Search Central. "How Google Search Works." developers.google.com
3. Google Search Central. "Sitemaps Overview." developers.google.com

4. Google Search Central. "Robots.txt Introduction." developers.google.com
5. IndexNow. "Protocol Overview." indexnow.org