Comprehensive Architectural Analysis of Console Observability and Logging Optimization in Screeps

The architecture of an automated system in Screeps demands an observability layer that is both computationally inexpensive and semantically rich. In an environment where the player's primary interaction with the world occurs through the execution of sandboxed JavaScript, the console is the fundamental telemetry interface. However, the standard implementation of console logging often presents significant challenges, ranging from performance degradation and heap memory pressure to interface-specific rendering inconsistencies. This report provides a definitive research synthesis on the best practices for architecting a logging system within the Screeps environment, specifically addressing the mitigation of tick-by-tick spam, the technical nuances of HTML and CSS formatting in the console, and the optimization of server-side CPU utilization through strategic telemetry design.

The Computational Mechanics of the Screeps Logging Pipeline

To establish a robust logging framework, it is first necessary to analyze the causal chain of data as it moves from the server-side execution environment to the client-side user interface. Screeps utilizes the isolated-vm library to run user code in a highly restricted, secure sandbox.[1] Every call to console.log() triggers a mechanism that captures the output during the script's execution phase in the game loop.[2] This process is not merely a visual artifact; it represents a data transmission event that occurs within the constraints of the game's tick-based architecture.

The lifecycle of a log entry begins with the instantiation of a string in the script's heap. In the V8 engine, string concatenation and manipulation are memory-intensive operations. When a player constructs complex log messages using standard string interpolation or the + operator, the engine must allocate new memory segments for these temporary objects. Research indicates that frequent memory allocations within the sandbox contribute to increased garbage collection frequency.[3] For advanced players running scripts at high CPU limits—up to 300 ms per tick for Global Control Level (GCL) 30 players—the cumulative cost of string processing can consume a non-trivial portion of the available CPU bucket.[4]

Pipeline Stage	Resource Constraint	Primary Risk
String Construction	Server CPU / Heap Memory	Garbage Collection spikes

		and timeout risks
Serialization	Server CPU	Recursive complexity in JSON.stringify() calls
Transmission	WebSocket Bandwidth	Connection throttling and rate limiting
DOM Rendering	Client CPU / Memory	Interface lag and Chromium memory leaks

The data points to a consistent trend where excessive logging overhead shifts from a server-side concern to a client-side bottleneck as the complexity of the AI increases.[6] While a simple string log might cost less than 0.01 CPU on the server, the cumulative effect of transmitting thousands of characters to the browser client creates a heavy load on the Document Object Model (DOM).[7] The official documentation and community feedback emphasize that the web version of the game forwards all console output to the browser's developer tools.[9] This duplication, while useful for deep object inspection, significantly multiplies the rendering overhead, often causing the simulation mode to slow to a crawl.[7]

Strategic Throttling and Signal-to-Noise Ratio Optimization

The most frequent complaint among Screeps developers is the "spam" effect, where routine operations flood the console every tick, making it impossible to identify critical alerts. Effective logging must distinguish between "telemetry" (data meant for automated analysis) and "notifications" (data meant for human observation).

Modulo-Based Temporal Throttling

The standard mechanism for reducing log volume is the implementation of periodic logging using the Game.time property.[2] By applying the modulo operator, a script can ensure that status updates are emitted only once every N ticks. This effectively reduces the logging frequency from a per-tick basis to a manageable interval. However, a naive implementation where multiple systems all log on Game.time % 100 == 0 creates a synchronized CPU spike every 100 ticks.[11]

To achieve a smoother performance profile, the analysis suggests the use of staggered offsets.[12] By adding a unique identifier or a random salt to the Game.time before the modulo operation, the logging events are distributed across the cycle. For example, logging a room's energy status on (Game.time + roomNameHash) % 100 == 0 ensures that different rooms report their status at different points in the 100-tick cycle, flattening the CPU load and maintaining a steady stream of information rather than periodic bursts.

Log Frequency	Strategic Intent	Implementation Pattern
Per Tick	Critical Errors / Attack Alerts	if (isCritical) { log(); }
10-20 Ticks	Operational State Transitions	if (stateChanged) { log(); }
100 Ticks	Resource Efficiency Stats	if (Game.time % 100 == 0) { log(); }
500-1000 Ticks	Long-term GCL/RCL Projections	if (Game.time % 500 == 0) { log(); }

State-Transition and Delta Logging

A more advanced strategy for telemetry is delta logging, where information is emitted only when a significant change in the game state occurs.[13] For instance, instead of logging the hits of a rampart every tick, the system should log only when the hits cross a certain threshold or when the rampart is actively being repaired by a tower. This approach requires maintaining a local cache of the "last logged state" within the Memory object or the heap.[15]

Causal relationships in the data suggest that delta logging is far more effective for complex bots managing multiple rooms.[17] As the empire expands, the sheer number of objects makes periodic logging of every unit unfeasible. By focusing on state changes—such as a creep transitioning from HARVESTING to UPGRADING—the developer can reconstruct the entire timeline of events from a fraction of the data required by tick-based logging.[18]

Technical Troubleshooting of HTML and CSS Formatting

The user has identified persistent issues with the rendering of <font> and <span> tags, while noting that emojis function without fail. This discrepancy is rooted in the way different game clients parse and sanitize the HTML stream emitted by the console.log() command.[19]

The Role of HTML Sanitization and Security

Since the inception of the project, the Screeps admins have allowed console.log() to accept HTML for the purpose of creating prettier console outputs.[21] This historical decision led to the "Client Abuse" community, where players used <script> and <style> tags to inject custom functionality into the official clients.[21] To prevent security vulnerabilities such as cross-site scripting (XSS), the game client now utilizes a sanitization layer.[22]

Tags like and <font> are generally allowed, but their attributes are heavily filtered. The failure of tags usually occurs when the style attribute contains properties that the sanitizer deems unsafe, such as position: fixed or width: 100%, which could be used to obscure the main game UI.[25] Furthermore, renderers in different environments (such as the Steam client vs. a standard browser) have varying support for modern CSS properties.[27]

Formatting Method	Technical Support	Primary Limitation
<font color="...">	Universal (Web/Steam/Terminal)	Deprecated; limited to color/size
	High (Web/Modern Steam)	Strict sanitization of CSS properties
<b> / <i> / <u>	Universal	Limited semantic range
Emojis (Unicode)	Universal	OS-dependent font availability

Optimizing CSS Precedence and Syntax

If formatting tags are being ignored, the analysis points to three likely culprits:

1. **Improper Nesting and Quotation:** The Screeps console parser is often less forgiving than a standard web browser. Research into similar environments suggests that the lack of quotes around attribute values or the use of single quotes when double quotes are expected can cause the parser to fail.[29] The safest pattern is to use escaped double quotes for internal attributes: console.log("...");.
2. **CSS Specificity and Overrides:** The game client's own stylesheet often has global rules for console text. If a player uses a <span> tag, its properties may be overwritten by the client's default font settings unless the style is declared with sufficient specificity or defined inline.[31]
3. **The "Inherit" Pattern:** In some browser versions, the console environment does not

automatically inherit the parent's text properties for inline elements. Using all: inherit or color: inherit can sometimes fix transparency or layout issues in formatted logs.[34]

The Resilience of Unicode Iconography

The reason emojis work reliably is that they are not part of the HTML processing layer; they are standard Unicode characters rendered by the client's system fonts.[27] In a professional observability context, emojis should be used as "visual tags" that allow for rapid scanning of logs. Emojis bypass the overhead of HTML parsing and are interpreted by the text server as single glyphs, making them the most CPU-efficient way to add color and categorization to the console.[37]

Architectural Best Practices for a Logging Utility

Rather than invoking console.log() directly throughout the codebase, it is standard practice among high-level Screeps developers to implement a dedicated Logger class.[39] This abstraction layer provides several critical advantages for large-scale empire management.

Centralized Level Filtering

A robust logger implements severities (ERROR, WARN, INFO, DEBUG, TRACE).[41] This allows the user to change the global verbosity level through a single memory edit without redeploying code. During normal operation, the level is set to INFO, suppressing the "noise" of creep actions. During active debugging or defense, the level can be dropped to DEBUG to provide more granular visibility into the AI's logic.[41]

Level	Severity	Visual Signal	Operational Trigger
ERROR	5	🔴 Red / 💥	Script crash, spawn failure, invasion detected
WARN	4	🟡 Yellow / ⚠️	Energy shortage, construction stagnation
INFO	3	⚪ White / ℹ️	RCL/GCL upgrades, market transactions
DEBUG	2	🔵 Blue / 🔍	Decision-making logic, pathfinding

			costs
TRACE	1	🟣 Purple / 📝	Individual creep intents and task updates

Decoupling Logic from Output

The analysis of professional repositories like ScreepsDotNet and the Overmind bot reveals a pattern of decoupling the "what" from the "where".[39] A well-structured logger should be able to route output to different sinks. While the console is the primary sink, higher-order telemetry can be written to memory segments for ingestion by external tools like Grafana or Kibana via screeps-stats.[46] This allows the console to remain clean for the human operator while maintaining a complete historical record of the AI's performance in a specialized analytics environment.

Performance Through Minimal Evaluation

A significant source of "lost" CPU is the evaluation of log strings that are ultimately suppressed by the current log level. A professional logger should check the level *before* constructing the string.

JavaScript

```javascript
// Optimized Pattern (Narrative Description)
// Instead of:
// log.debug("Creep " + name + " is at " + pos);
// Use:
// if (log.level <= DEBUG) { log.emit("Creep " + name + " is at " + pos); }
```

By wrapping the logging call in a level check, the script avoids the expensive concatenation and serialization of data that will never be displayed.[43] This is particularly critical for objects being serialized via JSON.stringify(), as the performance cost can be hundreds of times higher than a standard property lookup.[8]

Client-Specific Formatting Nuances

The Screeps ecosystem is served by multiple clients, each with distinct rendering engines. Best practices for logging must account for the common denominators and the specific advantages of each platform.

The Official Web Client

The web client is based on standard Chromium components. It provides the best support for room links using the syntax W1N1.[50] These links are interactive elements that shift the game's focus to the specified room. The data suggests that making room names interactive in logs is one of the most effective ways to improve base management efficiency.[19]

The Steam Desktop Client

The Steam client is an Electron application that reuses the web code but operates in a different security context. Causal analysis of historical vulnerabilities shows that the Steam client previously lacked some of the sandboxing present in web browsers, leading to risks where maliciously crafted HTML in a player's sign could execute commands on a victim's machine.[21] Modern updates have mitigated this through stricter sanitization and updated Chromium versions.[27] However, the Steam client remains more prone to memory bloat from console spam than a standalone Chrome window.[10]

Terminal Clients (Multimeter and Screeps-Console)

For advanced users, terminal clients like screeps-multimeter provide a lightweight alternative to the graphical UI.[20] These clients do not render HTML; instead, they translate a subset of tags into ANSI terminal colors.[57] To ensure logs are readable in these tools:

- Use <font color="..."> for basic coloring, as most terminal wrappers are hardcoded to support it.[57]
- Avoid complex inline CSS in <span> tags, as they will often be stripped out entirely, leaving plain text.[57]
- Ensure that the color hex codes are 6-character strings (e.g., #FF0000) rather than 3-character shorthand, which some basic terminal renderers fail to parse.[58]

Memory Management and Persistence in Telemetry

Logging in Screeps is deeply intertwined with the Memory object and the concept of "Global Resets." Understanding these relationships is critical for accurate troubleshooting.

The Impact of Global Resets

Global resets occur when the game engine migrates a player's script to a different server node or when the user uploads new code.[15] During a reset, all heap variables (those not stored in Memory) are wiped. A professional logging system must log the occurrence of a global reset to alert the operator that any non-persisted caches have been invalidated.[15] This is often the time when "startup" logs are most critical, providing a snapshot of the AI's initialization parameters.[41]

Managing Memory Bloat

While it may be tempting to store a history of logs in the Memory object, this is highly discouraged. The Memory object must be serialized and deserialized every tick, and its CPU cost is directly proportional to its size.[3] Storing thousands of log strings in Memory.log will rapidly inflate the AI's baseline CPU usage, potentially leading to timeouts.[51] Instead, logs should be treated as ephemeral streaming data, while historical stats should be aggregated into numerical counters or stored in memory segments, which are only loaded on demand.[47]

Persitence Method	**CPU Cost**	**Use Case**
Console Stream	Minimal	Real-time human monitoring
Memory Storage	Very High	Critical state persistence (not logs)
Memory Segments	High (on load)	Historical analytics and stat dumps
Game.notify()	Minimal (Intent)	Out-of-game emergency alerts

Advanced Observability and Profiling Techniques

For the most complex Screeps repositories, such as the one at https://github.com/grgisme/screeps, simple logging is often insufficient. Advanced developers leverage profiling tools and error mapping to diagnose performance issues and logical failures.

Error Mapping and Transpilation

If the codebase utilizes TypeScript or a bundler like Webpack/Rollup, the standard stack traces in the console will point to a single compiled file (e.g., main.js), rendering line numbers useless.[49] Implementing an ErrorMapper utility is essential. These utilities use source maps to translate the compiled stack trace back to the original source files, allowing the console to output accurate file paths and line numbers.[49] This transformation is usually performed within a try-catch block at the entry point of the main loop.[19]

The Screeps Profiler

The community-developed screeps-profiler is the gold standard for performance logging.[12] It works by wrapping standard game prototypes (like Creep, Room, and Spawn) and measuring the CPU used by each method call.[12] The profiler then outputs a formatted table to the console, showing the average and total CPU usage for every function in the AI. This is the ultimate tool for identifying "invisible" CPU leaks that logging alone cannot catch.

Security Considerations and Safe Logging

Because console.log() parses HTML, it creates a potential vulnerability if the script logs untrusted data from other players.[21] This includes creep names, room signs, and public memory segments.

Sanitizing Untrusted Inputs

A malicious player could name a creep something like <script>localStorage.auth =...</script>. If a victim's script finds this creep and logs its name via console.log(creep.name), the script will execute in the victim's browser, potentially stealing their authentication token.[21]

To prevent this, any data originated from an external source must be sanitized or escaped.[26] Simple regex-based escaping of < and > characters is a baseline requirement for professional-grade bots.[72]

JavaScript

```
// Safe Logging Pattern (Narrative)
// let safeName = untrustedData.replace(/</g, "<").replace(/>/g, ">");
// log.emit("Encountered creep: " + safeName);
```

This prevents the browser from interpreting the logged data as HTML tags, effectively neutralizing any "Client Abuse" attempts while still displaying the relevant information.[26]

Conclusions and Practical Implementation Path

The evidence gathered through extensive research into Screeps console logging suggests that an optimal observability strategy is founded on three pillars: performance, clarity, and cross-client compatibility.

First, the logging system must be performance-aware. The use of a severity-based global logger ensures that expensive string manipulations are only performed when necessary.

Developers should leverage the browser's developer tools for deep object inspection rather than using JSON.stringify() for high-frequency logs.

Second, clarity is achieved through intelligent throttling and semantic formatting. Using Game.time with staggered offsets prevents simultaneous CPU spikes, while state-transition logging provides a high signal-to-noise ratio. Unicode iconography (emojis) should be prioritized for cross-client visual signals, as it bypasses the complexities of HTML sanitization and CSS overrides.

Third, the formatting logic must be client-agnostic. While <span> tags with inline styles are the modern standard, ensuring basic compatibility with <font color="..."> is essential for those who use terminal-based third-party tools.

For the operator managing the repository at https://github.com/grgisme/screeps, the immediate priority should be the implementation of a centralized logging utility that wraps the entry points with an ErrorMapper and utilizes a severity filter. This will transform the console from a source of spam and client-side lag into a precision tool for tactical and strategic oversight. The transition from per-tick output to event-driven telemetry is the hallmark of a mature Screeps AI, allowing for continued expansion without compromising the stability of the observability layer. By adhering to these structural patterns, the operator can ensure that their automated empire remains both visible and efficient, even as it scales to manage dozens of rooms across multiple shards.

Works cited

1. Building a LeetCode-style code evaluator with isolated-vm - LogRocket Blog, accessed February 17, 2026, https://blog.logrocket.com/building-leetcode-style-code-evaluator-isolated-vm/
2. Understanding game loop, time and ticks - Screeps Documentation, accessed February 17, 2026, https://docs.screeps.com/game-loop.html
3. Need for Speed - Endgame optimization limitations | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/298/need-for-speed-endgame-optimization-limitations
4. Screeps on Steam - cs.wisc.edu, accessed February 17, 2026, https://pages.cs.wisc.edu/~linzuo/deliverables/stage1/steam/steam785.html?l=norwegian
5. Control | Screeps Documentation, accessed February 17, 2026, https://docs.screeps.com/control.html
6. Does calling console.log use CPU? :: Screeps: World Help - Steam Community, accessed February 17, 2026, https://steamcommunity.com/app/464350/discussions/5/351660338715419646/?l=french
7. Simulator gets slower and slower | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/2181/simulator-gets-slower-and-slower

8. Screeps: write debug output to console? - javascript - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/27070203/screeps-write-debug-output-to-console
9. Debugging | Screeps Documentation, accessed February 17, 2026, https://docs.screeps.com/debugging.html
10. Which browser/plugin/embed does the steam client use? :: Screeps: World 综合讨论, accessed February 17, 2026, https://steamcommunity.com/app/464350/discussions/0/1470840994964782503/?l=schinese
11. New Player, Newb Coder, Dozens of Questions : r/screeps - Reddit, accessed February 17, 2026, https://www.reddit.com/r/screeps/comments/53uv4u/new_player_newb_coder_dozens_of_questions/
12. Need some help improving my CPU usage | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/2381/need-some-help-improving-my-cpu-usage
13. Workflow tips and prioritization for new players? | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/2556/workflow-tips-and-prioritization-for-new-players
14. How much logging is too much? (ASP.NET) : r/ExperiencedDevs - Reddit, accessed February 17, 2026, https://www.reddit.com/r/ExperiencedDevs/comments/1jt6o3u/how_much_logging_is_too_much_aspnet/
15. OOP Ideas for Screeps/JS, accessed February 17, 2026, https://screeps.com/forum/topic/2777/oop-ideas-for-screeps-js
16. My creep spawning code doesn't work | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/2533/my-creep-spawning-code-doesn-t-work
17. tanjera/screeps: Scripts for my Screeps MMO colonies (JS; ongoing). - GitHub, accessed February 17, 2026, https://github.com/tanjera/screeps
18. Tutorial 1.1 : Creeps, Roles, Actions, Tasks, Jobs, and Priorities | Screeping, accessed February 17, 2026, https://screeping.wordpress.com/2020/06/22/tutorial-1-1-creeps-roles-actions-tasks-jobs-and-priorities/
19. Basic Debugging - Screeps Wiki, accessed February 17, 2026, https://wiki.screepspl.us/Basic_debugging/
20. screeps-multimeter/README.md at master - GitHub, accessed February 17, 2026, https://github.com/screepers/screeps-multimeter/blob/master/README.md
21. Client Abuse - Screeps Wiki, accessed February 17, 2026, https://wiki.screepspl.us/Client_Abuse/
22. Supported HTML tags in Source Code editor - PlayPosit Knowledge, accessed February 17, 2026,

https://knowledge.playposit.com/article/252-supported-html-tags-in-rte
23. Default TAGs ATTRIBUTEs allow list & blocklist · cure53/DOMPurify Wiki - GitHub, accessed February 17, 2026, https://github.com/cure53/DOMPurify/wiki/Default-TAGs-ATTRIBUTEs-allow-list-&-blocklist
24. To Configure Allowed HTML Tags and Attributes - Ivanti, accessed February 17, 2026, https://help.ivanti.com/ht/help/en_US/ISM/2025/admin-user/Content/Configure/SetUpWizard/Configure%20Allowed%20Tags%20and%20Attribute.htm
25. Content-Security-Policy: style-src directive - HTTP - MDN - Mozilla, accessed February 17, 2026, https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy/style-src
26. Why is a
27. Displaying Text with Say and Room.Visual [Fixed] :: Screeps: World General Discussions, accessed February 17, 2026, https://steamcommunity.com/app/464350/discussions/0/1473095965295570453/
28. How can I prevent having just one hanging word on a new line in an HTML element?, accessed February 17, 2026, https://stackoverflow.com/questions/31974448/how-can-i-prevent-having-just-one-hanging-word-on-a-new-line-in-an-html-element
29. Why aren't my tags working? - Codecademy, accessed February 17, 2026, https://www.codecademy.com/forum_questions/517b0506d2b8c1b78600180c
30. Top HTML CSS JAVASCRIPT Interview Questions & Answers 2026 - H2K Infosys, accessed February 17, 2026, https://www.h2kinfosys.com/blog/top-html-css-javascript-interview-questions-answers/
31. Using
32. span class id isnt working (Example) | Treehouse Community, accessed February 17, 2026, https://teamtreehouse.com/community/span-class-id-isnt-working
33. CSS span formatting - html - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/21933968/css-span-formatting
34. Format console.log with color and variables surrounding non-formatted text - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/41898612/format-console-log-with-color-and-variables-surrounding-non-formatted-text
35. Different font-size to span tag inside of p - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/23338649/different-font-size-to-span-tag-inside-of-p
36. rhoit/dot-emacs: my emacs config :godmode - GitHub, accessed February 17, 2026, https://github.com/rhoit/dot-emacs
37. TextServer — Godot Engine (stable) documentation in English, accessed February 17, 2026, https://docs.godotengine.org/en/stable/classes/class_textserver.html
38. Game - Screeps Documentation, accessed February 17, 2026,

https://docs.screeps.com/api/
39. accessed December 31, 1969, https://github.com/overmind-screeps/Overmind/blob/master/src/console/Console.ts
40. accessed December 31, 1969, https://raw.githubusercontent.com/overmind-screeps/Overmind/master/src/console/Console.ts
41. When to use the different log levels - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/2031163/when-to-use-the-different-log-levels
42. Log Debug vs. Info vs. Warn vs. Error: Understanding Log Levels - EdgeDelta, accessed February 17, 2026, https://edgedelta.com/company/blog/log-debug-vs-info-vs-warn-vs-error-and-fatal
43. Logging levels explained. Good old logs remain a very important… | by Michael Merkulov | Medium, accessed February 17, 2026, https://medium.com/@m.merkulov/logging-levels-explained-4dd61815601f
44. 10 Essential Logging Best Practices for Developers - Kloudfuse, accessed February 17, 2026, https://www.kloudfuse.com/blog/logging-best-practices-for-developers
45. thomasfn/ScreepsDotNet: Tools to support writing bots for Screeps Arena and Screeps World using .Net 8.0. - GitHub, accessed February 17, 2026, https://github.com/thomasfn/ScreepsDotNet
46. A client-side library to enhance the output of screeps-stats statistics. - GitHub, accessed February 17, 2026, https://github.com/screepers/screeps-stats-lib
47. Third Party Tools | Screeps Documentation, accessed February 17, 2026, https://docs.screeps.com/third-party.html
48. What tips do you have for logging? And by logging, I mean really good logging. - Reddit, accessed February 17, 2026, https://www.reddit.com/r/webdev/comments/tj1f26/what_tips_do_you_have_for_logging_and_by_logging/
49. shup1/xxscreeps - NPM, accessed February 17, 2026, https://www.npmjs.com/package/%40shup1%2Fxxscreeps
50. Steam client > Console link to room? - forum - Screeps, accessed February 17, 2026, https://screeps.com/forum/topic/763/steam-client-console-link-to-room
51. justjavac/screeps-tips: Tips of the day for Screeps Game. - GitHub, accessed February 17, 2026, https://github.com/justjavac/screeps-tips
52. Screeps: How A Game About Programming Sold Its Players a Remote Access Trojan, accessed February 17, 2026, https://outsidetheasylum.blog/screeps/
53. WebGL optimization | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/2658/webgl-optimization
54. Open source alternative? · Issue #53 · screeps/screeps - GitHub, accessed February 17, 2026, https://github.com/screeps/screeps/issues/53
55. screepers/screeps-multimeter: The most useful tool on your screeps workbench.

- GitHub, accessed February 17, 2026, https://github.com/screepers/screeps-multimeter
56. screeps-multimeter - NPM, accessed February 17, 2026, https://www.npmjs.com/package/screeps-multimeter
57. screepers/screeps_console: Standalone Interactive Screeps Console - GitHub, accessed February 17, 2026, https://github.com/screepers/screeps_console
58. How to validate Hexadecimal Color Code using Regular Expression - GeeksforGeeks, accessed February 17, 2026, https://www.geeksforgeeks.org/dsa/how-to-validate-hexadecimal-color-code-using-regular-expression/
59. Colorizing text in the console with C++ - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/4053837/colorizing-text-in-the-console-with-c
60. Category: Changelogs | Screeps Blog, accessed February 17, 2026, https://blog.screeps.com/categories/Changelogs/
61. Mysterious error without stack trace | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/1718/mysterious-error-without-stack-trace
62. Making your own console functions - Screeps Wiki, accessed February 17, 2026, https://wiki.screepspl.us/Making_your_own_console_functions/
63. accessed December 31, 1969, https://raw.githubusercontent.com/grgisme/screeps/master/package.json
64. Auth Tokens - forum - Screeps, accessed February 17, 2026, https://screeps.com/forum/topic/2052/auth-tokens
65. Pushing code to Screeps, accessed February 17, 2026, https://wiki.screepspl.us/Pushing_code_to_Screeps/
66. accessed December 31, 1969, https://github.com/screepers/screeps-typescript-starter/blob/master/src/utils/ErrorMapper.ts
67. accessed December 31, 1969, https://raw.githubusercontent.com/screepers/screeps-typescript-starter/master/src/utils/ErrorMapper.ts
68. [SOLVED] Can't Operate Console When Experiencing Error | Screeps Forum, accessed February 17, 2026, https://screeps.com/forum/topic/119/solved-can-t-operate-console-when-experiencing-error
69. How do I know why a function is using that much CPU ? : r/screeps - Reddit, accessed February 17, 2026, https://www.reddit.com/r/screeps/comments/7rzczp/how_do_i_know_why_a_function_is_using_that_much/
70. How can I allow my user to insert HTML code, without risks? (not only technical risks), accessed February 17, 2026, https://stackoverflow.com/questions/701580/how-can-i-allow-my-user-to-insert-html-code-without-risks-not-only-technical

71. HTML_WHITELIST Function - Oracle Help Center, accessed February 17, 2026, https://docs.oracle.com/en/database/oracle/application-express/18.2/aeapi/HTML_WHITELIST-Function.html
72. screeps - GitHub Gist, accessed February 17, 2026, https://gist.github.com/derofim/e19a57bfff7f3ae20de8
73. White list of HTML tags I should allow from user generated content? - Stack Overflow, accessed February 17, 2026, https://stackoverflow.com/questions/9185495/white-list-of-html-tags-i-should-allow-from-user-generated-content