METEHAN.AI RESEARCH REPORT · PREPRINT FIELD GUIDE

# Google Discover Telemetry: Field Glossary and Concept Map

**Metehan Yesilyurt**

*AI Search & Information Retrieval Researcher · metehan.ai*

https://metehan.ai/pdf/google-discover-telemetry-field-glossary.pdf

**Abstract.** A working glossary and decision map for the client-side vocabulary of Google Discover: parsing precedence, two-level filtering, NAIADES subtypes, cluster names, and feed delivery. Each entry states what the observable trace can and cannot prove, with rules and an audit sequence you can run on a single page.

**Keywords:** Google Discover, client telemetry, Schema.org JSON-LD, NAIADES, content filtering, field glossary

Companion to the article "Google Discover Architecture: Clusters, Classifiers, OG Tags, NAIADES -. The article covers the topic in full; this document is a separate, practical field glossary and concept map.

## 1. Concept Map: Eight Checkpoints, Not One Algorithm

I read the client-side picture of Discover as a chain of checkpoints rather than a single ranking machine. Each checkpoint has a name, a fixed position in the order of operations, and a boundary on what its evidence can support. The order carries more diagnostic value than any individual term: ingestion, structured data and Open Graph parsing, classification, filtering, interest matching, server-side ranking, feed assembly, and the feedback loop [1].

One ordering fact changes how I read everything downstream. Collection-level filtering runs before interest matching and before ranking [1]. A domain stopped at collection scope never competes for a slot, so improving a single URL's relevance cannot rescue it. When someone asks me why a page vanished, my first question is never about content quality. It is about which checkpoint the page failed to pass.

| Checkpoint | Observable trace | Glossary anchor | Test I run |
|---|---|---|---|
| Ingestion | Knowledge Graph MIDs on recognized topics | Entity extraction | Check whether the topic resolves to a MID |
| Parsing | Hardcoded tag precedence order | Fallback chain | Strip JSON-LD and see which fields survive |
| Paywall classification | Content tier strings plus warning event 38468 | Content tier | Count tier values per page |
| Filtering | filter_collection_status and filter_entity_status counters | Two-level filtering | Separate domain scope from URL scope |
| Interest matching | NAIADES subtype enum values | Personalization subtype | Verify publisher registration status |
| Ranking | Nothing observable from the client | Server-side boundary | Refuse to claim a client string ranks pages |
| Assembly | gRPC service names, streaming renderer | Live feed | Watch for mid-session reordering |
| Feedback | Tombstone boolean on the content state object | Tombstoning | Assume dismissals are permanent |

## 2. Parsing Layer: Precedence Terms You Will Actually Use

A fallback chain is an ordered list the parser walks until it finds a value. Once an earlier source supplies the field, later sources are never consulted. This is described in the source analysis as hardcoded behavior rather than preference [1], which is why adding tags rarely changes the outcome: you have to change which tag wins.

| Field | Order checked |
|---|---|
| Title | Schema.org JSON-LD, then og:title, then twitter:title, then the generic title tag |
| Author | Schema.org JSON-LD, then the author name attribute |
| Publisher | Schema.org JSON-LD, then og:site_name |
| Image | og:image, then twitter:image, then og:image:secure_url, then twitter:image:src, then a generic image tag |
| Language | Execution-based primary language detection, then og:locale, then inLanguage in JSON-LD, then a hardcoded default of en gated behind a server config flag |

Worked example. A template emits a JSON-LD headline, an og:title that differs, and a twitter:title that differs again. Only the JSON-LD string is packaged for title. If I want the Open Graph string used, I must remove or align the JSON-LD field. Adding a fourth tag does nothing, because the chain stops at the first hit.

Content tier is the paywall label. The parser first reads isAccessibleForFree in JSON-LD, which defaults to true when absent, then reads article:content_tier with exactly three recognized values: free as the expected default, metered counted as paywalled, and locked counted as paywalled [1]. More than one content tier value on a page logs a warning with event code 38468 [1]. Two meta tags halt processing entirely with an exception: nopagereadaloud produces a readout disallow state and notranslate produces a translation disallow state [1].

- Rule: treat JSON-LD as the primary path and Open Graph as insurance, never the reverse, for title, author, and publisher [1].
- Rule: if a CMS plugin injects notranslate as a meta tag, treat it as a pipeline defect, not a style choice [1].
- Rule: allow exactly one article:content_tier value per page [1].
- Caution: an earlier claim that exactly six Open Graph tags are parsed was corrected in the source, which notes the parser also handles twitter:image, twitter:title, twitter:image:src, author, title, inLanguage, isAccessibleForFree, a generic image tag, and JSON-LD [1].

## 3. Filter Layer: Collection Scope Versus Entity Scope

Discover tracks two separate filter counters under the Discover streamz path, one for collection status and one for entity status, each parameterized by a reason value [1]. Collection scope targets a publisher or domain. Entity scope targets a single URL. In the card menu, choosing not to see content from a publisher triggers the collection-level path, and that reaction applies to all content from the domain rather than the article that provoked it [1].

| Term | Scope | Where it appears | What it cannot tell me |
|---|---|---|---|
| Collection filter | Publisher or domain | filter_collection_status with a reason parameter | Server-side thresholds and any recovery path |
| Entity filter | Single URL | filter_entity_status with a reason parameter | Which user action maps to which reason value |
| Tombstone | Single content item | Boolean field on the content state object | Whether reinstatement is ever possible; the source states dismissed content does not resurface [1] |
| Lifecycle states | Single content item | stallState, lastKnownState with INSERTED, REMOVED, or UNKNOWN, and purged | Whether a stall is temporary or terminal |
| Rug pull | Feed session | background_refresh_rug_pull_count | Why the content was withdrawn after delivery |

Decision rule for a disappearing URL: check sibling URLs from the same domain first. If the whole domain dropped, I diagnose at collection scope. If only one URL dropped, entity scope or tombstoning is the stronger hypothesis. The rug pull counter adds a third case: content that reached the feed and was then removed during a background refresh, which means withdrawal can happen after display, not only before it [1].

Counterfactual states deserve their own label. SHOW_SKIPPED_DUE_TO_COUNTERFACTUAL marks content withheld for experiment measurement, and VISIBILITY_REPRESSED_COUNTERFACTUAL marks elements deliberately suppressed for measurement, but the latter is a Google-wide visual element logging state rather than a Discover-specific one [1]. That distinction matters: one is a Discover effect, the other is a measurement artifact you can observe across the app.

## 4. Personalization Layer: The NAIADES Subtype Map

NAIADES appears as a personalization system with enumerated content subtypes, all confirmed as enum values in the source analysis [1]. I treat these as category labels, not weights. The label tells me what kind of signal a content item carries; it does not tell me how much that signal counts.

| Subtype | Enum | How I read it |
|---|---|---|
| SUBTYPE_PERSONAL_UPDATE_MID_BASED_NAIADES | 793 | Entity or Knowledge Graph based personalization |
| SUBTYPE_PERSONAL_UPDATE_QUERY_BASED_NAIADES | 792 | Search query based personalization |
| SUBTYPE_PERSONAL_UPDATE_QUERY_BASED_NAIADES_PERSISTENT_LOGGING | 805 | Same category with persistent logging |
| SUBTYPE_PERSONAL_UPDATE_RECALL_BOOST | 797 | Increased retrieval priority from the candidate pool, before ranking |
| SUBTYPE_PERSONAL_UPDATE_WPAS | 800 | Web Publisher Articles Signal, likely tied to Publisher Center registration |
| SUBTYPE_PERSONAL_UPDATE_WPAS_PERSISTENT_LOGGING | 811 | Same category with persistent logging |
| SUBTYPE_PERSONAL_UPDATE_AIM_THREAD_NAIADES | 856 | AI Mode thread based personalization |

The practical consequence is experimental hygiene. If a publisher is registered in Publisher Center, the WPAS subtype is a separate variable from content quality, and I want it isolated before I attribute a performance change to anything else [1]. The recall boost subtype suggests retrieval priority is raised before ranking rather than during it, but the source is explicit that the weight attached to any subtype is a server-side decision that client code does not expose [1].

## 5. Surface Layer: Clusters, Beacon, and the Streaming Feed

Cluster names describe the slot a card occupies, not the merit of the article inside it. The observable cluster types include the primary content cluster, location-based stories, trending topic narratives with a companion variant, recent video, a priority queue, breaking news headlines, widget cards such as weather and sports scores, related article groupings, trending user-generated content, sign-in prompts, cross-platform promotion, and an internal-codename cluster [1].

- neoncluster: the primary content cluster.
- geotargetingstories: location-based stories.
- deeptrends and deeptrendsfable: trending topic narratives.
- freshvideos: recent video content.
- mustntmiss: a priority queue of content the system treats as essential to show.
- newsstoriesheadlines: breaking news.
- homestack: widget cards such as weather and sports scores.
- garamondrelatedarticlegrouping: separate articles combined under a shared topic heading.

- trendingugc, signinlure, iospromo, moonstone: user-generated trending content, sign-in prompts, cross-platform promotion, and an internal codename.

Beacon is the push channel. It handles exactly two content types: sports scores at ordinal 0 and investment or finance recaps at ordinal 1, with anything else rejected as unsupported [1]. Sports content has more than ten dedicated notification types, while general breaking news has a single notification type [1]. The source flags this area as dynamic, so I record the observation with a date rather than treating it as fixed.

Delivery is a persistent connection rather than a static list. The observable gRPC surface includes a standard feed renderer with interactive query and next page calls, a streaming feed renderer with a streaming query, an actions service, a reactions service, a story recommendations service, and a homestack feed renderer [1]. The consequence for anyone measuring performance: the server can inject cards, reorder them, or remove stale content during an open session, so a single screenshot is not evidence of a stable feed.

## 6. Evidence Grading: How I Score a Telemetry Claim

Because every observation here comes from client code, I grade claims before I repeat them. The grading is what keeps a glossary useful instead of decorative.

| Grade | Evidence type | Example | How I use it |
|---|---|---|---|
| A | Exact string match in decompiled source | Enum values, streamz paths, gRPC service names | State as observed |
| B | Naming convention plus code proximity | WPAS read as a Publisher Center registration signal | State as inference |
| C | Borrowed from a shared system | VISIBILITY_REPRESSED_COUNTERFACTUAL is Google-wide; engagement_time_msec is a standard GA4 parameter | Do not attribute to Discover |
| D | Context mismatch | Freshness buckets appear in a gesture settings context; freshness_delta_in_seconds sits with Smartspace weather metrics | Do not present as Discover ranking inputs |
| E | Absent from this SDK version | PCTR_MODEL_TRIGGERED was not found | Absence is not disproof |

- Worked example one: EVERGREEN_VIBRANT is a UI color palette name in XgadsContext, alongside other vibrant palette names, not a content classification type [1].
- Worked example two: isCollectionHiddenFromEmberFeed concerns the Ember tab rather than the Discover feed, so it cannot support a claim about feed filtering [1].
- Worked example three: the freshness bucket strings exist as exact values, but their connection to Discover freshness scoring cannot be verified from the client alone [1].

Scale sets expectations. The analysis behind this vocabulary covered 87,498 classes across 13 DEX files, with 95.5 percent obfuscated under a single package and no ProGuard mapping available [1]. That is a large read but an incomplete one, so grade E items stay open on my list instead of being filed as false.

## 7. A Thirty Minute Page Audit

1. Fetch the raw HTML and list every title source: JSON-LD, og:title, twitter:title, and the generic title tag. Align them so the JSON-LD value is the one you want used [1].
2. Confirm that author and publisher exist in JSON-LD, because Open Graph is only the fallback for those two fields [1].
3. Walk the image chain in order and confirm each candidate loads. The chain runs five levels deep, and images at least 1200 pixels wide are the guidance for hero card eligibility [1].
4. Search templates for nopagereadaloud and notranslate meta tags. Either one throws an exception that halts processing for that page [1].
5. Count article:content_tier values and reduce them to one, chosen from free, metered, or locked [1].
6. Set isAccessibleForFree explicitly in JSON-LD on free pages, since the value defaults to true when the field is absent [1].
7. Register in Publisher Center and record WPAS status as a separate variable in any before and after comparison [1].
8. Before changing anything, decide the failure scope: a domain-wide drop points at collection filtering, a single URL drop points at entity filtering or tombstoning [1].
9. Write the observation date next to every finding, because server-side configuration can change without a client update [1].

## 8. Questions the Client Cannot Answer

I keep a standing list of gaps so that speculation stays labeled. These are the questions the observable instrumentation raises but does not settle.

- What thresholds trigger collection-level filtering, and what does recovery look like once a domain is affected?
- How do reason values map back to specific user actions?
- How much weight does each NAIADES subtype carry during interest matching?
- Does og:title influence ranking directly, or does it only inform server-side decisions as part of the packaged metadata?
- Is every rug pull tied to an experiment, or can withdrawal happen for other reasons?
- Where are freshness boundaries actually applied, given that the bucket strings were found outside a confirmed Discover class?

The line between observation and speculation is the most useful artifact in this whole vocabulary. Terms change, weights move, and server-side systems shift without a client release, so the durable skill is knowing which checkpoint produced a signal, what that signal can prove, and where the evidence stops. Machine-readable versions of the underlying analysis are available for anyone who wants to check a specific string rather than trust a summary [2][3], and the schema placement guidance is the natural next step once parsing precedence is understood [4].

## REFERENCES & SOURCES

[1] **Source analysis: Google Discover architecture and client telemetry**, https://metehan.ai/blog/google-discover-architecture/

[2] **Source analysis as JSON**, https://metehan.ai/api/post/google-discover-architecture.json

[3] **Facts with sources and dates**, https://metehan.ai/api/facts.json

[4] **Where to place schema code on a website**, https://metehan.ai/articles/where-to-put-schema-code-in-website-best-practices-for-modern-seo-and-ai-discovery/

[5] **AI Mode rank trackers**, https://metehan.ai/articles/best-ai-mode-rank-tracker/

[6] **Agent endpoint index**, https://metehan.ai/llms.txt