Conversion Trace API

Diese API liefert Einblick in einzelne Conversion-Tracking-Vorgänge ("Traces") – jeder eingehende Tracking-Aufruf (Bestellabschluss) wird mit allen Details protokolliert, egal ob er erfolgreich einer Kampagne zugeordnet werden konnte oder nicht. Diese Doku richtet sich an alle, die selbst gegen die API sprechen wollen (Scripting, eigene Auswertungen, Support-Tools).

Endpunkt & Request-Format

URL: `POST https://SUBDOMAIN/ws/admin/JSON/`

Content-Type: `application/x-www-form-urlencoded`

Der eigentliche Request steckt als JSON-String im Formularfeld `REQUEST`, dazu kommt `NETWORKID` als eigenes Feld:

Feld	Beschreibung
`NETWORKID`	Ihre Netzwerk-ID
`REQUEST`	URL-encodeter JSON-String, siehe unten

Aufbau von `REQUEST`

```
{
  "function": { "class": "orders", "method": "getOrderTraces" },
  "params": [
    { "limit": 50, "offset": 0, "sort": "timestamp", "sort_dir": "DESC" },
    { "id": "LOGIN_ID", "type": "adm", "network_id": "NETWORK_ID" }
  ],
  "token": "IHR_API_TOKEN",
  "login_id": "LOGIN_ID"
}
```

Key	Beschreibung
`function.class`	Immer `orders` für alle hier beschriebenen Methoden.
`function.method`	Name der aufzurufenden Methode, siehe Methoden.

Key	Beschreibung
`params[0]`	Die eigentlichen Filter/Argumente der jeweiligen Methode.
`params[1]`	Ihr Login-Kontext: `id` (Ihre Login-ID), `type` (i.d.R. `adm`), `network_id` (Ihre Netzwerk-ID).
`token` / `login_id`	Ihre Zugangsdaten, siehe Authentifizierung. Alternativ als Header `X-Auth-Token` / `X-Auth-ID` statt im Body.

Response-Format

Jede Antwort – Erfolg wie Fehler – kommt in derselben Hülle:

```
{
  "data": { "...": "Rückgabewert der Methode" },
  "log": "no logmessage provided",
  "sessionId": "",
  "session": null
}
```

Die eigentlichen Daten stehen immer im `data`-Feld.

Bei Fehlern (HTTP 500):

```
{ "error": true, "msg": "...", "type": "Ws\\Error", "data": [] }
```

Authentifizierung

Jeder Aufruf braucht ein gültiges Zugangsdaten-Paar: `login_id` + `token` (im `REQUEST`-JSON, siehe oben) oder alternativ als Header `X-Auth-ID` / `X-Auth-Token`.

Ihren API-Token erhalten Sie über Ihre Ansprechperson, oder – falls Sie bereits über einen klassischen Login verfügen – über den Login-Call: `function.class: "user"`, `method: "login"` mit `identifier`/`password`/`usertype`. Die Antwort enthält Ihre `login_id` sowie einen `token`, den Sie danach für alle weiteren Aufrufe wiederverwenden.

Methoden

`orders.getOrderTraces`

Gefilterte, sortierte, paginierte Liste der Traces.

Filter (`params[0]`):

Parameter	Typ	Beschreibung
`limit`	int	Default `50`, maximal `200`
`offset`	int	Default `0`
`order_id`	int	Exakter Treffer auf eine Bestell-ID
`campaign_id`	int	Exakter Treffer auf eine Kampagnen-ID
`trigger_id`	int	Exakter Treffer auf eine Trigger-ID
`ordertoken`	string	Teilstring-Suche im Ordertoken
`result`	string	Exakter Treffer auf `result` (siehe Result-Werte)
`order_created`	`"yes"` / `"1"` oder `"no"` / `"0"`	Filtert danach, ob aus dem Trace eine Bestellung entstanden ist
`date_from`	Datum/Zeitstempel	Traces ab diesem Zeitpunkt
`date_to`	Datum/Zeitstempel	Traces bis zu diesem Zeitpunkt (Tagesgrenze inklusive)
`matching_method`	string	Exakter Treffer auf die Zuordnungsmethode (z.B. `VoucherCode`)
`ip_hash`	string	Exakter Treffer auf den (gehashten) IP-Wert
`freetext`	string, mehrzeilig	Durchsucht Ordertoken, IP-Hash, Ergebnis-Detail und Fehlergrund gleichzeitig, sowie die Bestell-ID (exakt). Jede Zeile ist eine eigene Suche – mehrere Zeilen = mehrere IDs/Tokens auf einmal suchen.
`sort`	string	Eine von `timestamp`, `order_id`, `campaign_id`, `result`, `matching_method`, `ordertoken` (Default `timestamp`)
`sort_dir`	`ASC` / `DESC`	Default `DESC`

Response (`data`):

```
{
  "traces": [
    {
      "id": 123,
      "order_id": 456,
      "campaign_id": 78,
      "trigger_id": 9,
      "ordertoken": "abc123",
      "timestamp": "2026-07-06T10:00:00+02:00",
      "user_agent": "...",
      "ip_hash": "...",
      "matching_method": "VoucherCode",
      "matching_cascade": [ { "method": "VoucherCode", "matched": true, "reason": "...",
"details": {}, "timestamp": "..." } ],
      "patches_applied": [ { "type": "pre", "name": "...", "match_string": "...", "applied":
true, "changes": {} } ],
      "request_params": { "...": "alle GET/POST-Parameter des Trackingaufrufs" },
      "duplicate_check": { "constraint_key": "...", "is_duplicate": false, "is_freezed":
false, "detail": "..." },
      "device_info": { "device": "...", "device_type": "...", "os": "...", "browser": "...",
"browser_version": "..." },
      "connection_info": { "ip": "...", "referer": "...", "cookies": "..." },
      "result": "order_created",
      "result_detail": "...",
      "error_reason": null,
      "tracking_type": "client",
      "campaign_title": "Kampagnenname",
      "trigger_title": "Triggername"
    }
  ],
  "total": 1337,
  "limit": 50,
  "offset": 0
}
```


Die verschachtelten Felder (`matching_cascade`, `patches_applied`, `request_params`, `duplicate_check`, `device_info`, `connection_info`) kommen als echtes JSON-Objekt/Array zurück, nicht als String.

`orders.getOrderTrace`

Alle Traces zu **einer** Bestell-ID (chronologisch, neueste zuerst).

Params: `{ "id": 456 }`

Response: Array von Trace-Objekten (gleiches Format wie oben, ohne `campaign_title` / `trigger_title`).

`orders.getOrderTraceResults`

Liefert verfügbare Filterwerte (Result, Match Method, Campaign) inkl. Trefferzahl – praktisch, um eigene Filter-Dropdowns zu befüllen.

Params: keine.

Response:

```
{
  "results": [ { "result": "order_created", "count": 900 }, ... ],
  "matching_methods": [ { "matching_method": "VoucherCode", "count": 300 }, ... ],
  "campaigns": [ { "campaign_id": 78, "campaign_title": "...", "count": 120 }, ... ]
}
```

`orders.getOrderTraceStats`

Liefert aggregierte Kennzahlen (stündlicher Verlauf + Zähler). Akzeptiert dieselben Filter wie `getOrderTraces`, mit Ausnahme von `order_id` / `ordertoken` / `trigger_id` / `ip_hash` / `sort` / `sort_dir`.

Filter (`params[0]`): `campaign_id`, `result`, `matching_method`, `order_created`, `freetext`, `date_from`, `date_to` (gleiche Bedeutung wie bei `getOrderTraces`).

Response:

```
{
  "chart": [ { "hour": "2026-07-06T09:00:00+00:00", "total": 42, "success": 30, "errors": 2,
"no_match": 10 } ],
  "errors_1h": 2,
  "success_1h": 30,
  "no_match": 10
}
```

`chart` ist stündlich gruppiert. Ohne `date_from` / `date_to` beziehen sich `chart` auf die letzten 24h und die Zähler `errors_1h` / `success_1h` auf die letzte Stunde, `no_match` auf die letzten 24h. Sobald ein

Datumsfilter gesetzt ist, gilt dieser Zeitraum für alle drei.

`orders.retraceOrder`

Führt einen gespeicherten Trace **erneut aus** – ein echter Tracking-Aufruf mit den ursprünglichen Parametern (User-Agent, Cookies, IP, Referer werden nachgebildet).

Params: `{ "trace_id": 123 }`

Response (Auszug):

```
{
  "trace_id": 123,
  "type": "etrack",
  "http_code": 200,
  "total_time": "184ms",
  "request": { "method": "GET", "user_agent": "...", "headers": [...], "cookies": "...",
"params": {...} },
  "response": { "...": "Antwort des Tracking-Aufrufs" },
  "success": true,
  "order_id": 999,
  "original_trace": { "id": 123, "result": "order_created", "order_id": 456, "timestamp":
"...", "matching_method": "VoucherCode" }
}
```

> **Achtung:** Das ist kein Dry-Run. Bei erfolgreichem Retrace entsteht eine echte neue Bestellung im System.

`orders.retrackTags`

Spielt für angegebene Bestellungen serverseitige Tags gegen die aktuell hinterlegten Bedingungen neu durch. Zwei Aufrufvarianten:

Parameter	Beschreibung
`traces`	Array bereits geladener Trace-Objekte (z.B. aus einem vorherigen `getOrderTraces`-Aufruf)
`order_ids`	Alternativ: String mit Bestell-IDs, komma-/zeilen-/leerzeichen-getrennt

---

Result-Werte

Wert	Bedeutung
`order_created`	Bestellung wurde erfolgreich erstellt
`duplicate`	Doppelte Bestellung erkannt
`freezed`	Bestellung eingefroren (Schutz vor Mehrfachbuchung)
`no_match`	Keine passende Kampagne/Zuordnung gefunden (kein Fehler)
`error`	Technischer Fehler
`campaign_blocked`	Kampagne ist blockiert/pausiert

Fehlerbehandlung

Fehler kommen im Format `{ "error": true, "msg": "...", "type": "Ws\\Error", "data": [] }` mit HTTP-Status 500. Eine leere Ergebnismenge (z.B. Filter ohne Treffer) ist **kein** Fehler – hier kommt ganz normal `"data"` mit leerem `traces`-Array bzw. `"total": 0` zurück.

Revision #2
Created 7 July 2026 10:33:21 by Leonie Engstfeld
Updated 7 July 2026 10:39:19 by Leonie Engstfeld