REMIT CLOUD

www.remitcloud.de

Webware RRM Service API

Documentation

Version 2.1

Webware internet solutions GmbH
Teichstraße 14-16
34130 Kassel Germany

Contents

1. Overview

The RemitCloud API allows your systems to integrate directly with our reporting platform.

Main capabilities:

- **Upload XML files** to RemitCloud
- **Submit REMIT Table 1 / Table 2 transactions as JSON** (POST /reports) with full TRUM field mapping
- **Track processing status** per file and load_id
- **Retrieve file history and processing results** (RRM, ACER, and ELCOM responses)
- **Download stored XML files** by load_id

Your existing web interface remains unchanged; the API is an additional technical interface for automation.

See section 4 for a map of API identifiers (external_reference, remit_file_id, load_id, etc.) and how they relate across the JSON submission flow.

2. Authentication

Authentication method: Bearer token

Where to get the token:

In the RemitCloud **web interface**, on your **user profile → API Tokens** tab:

- Create new API tokens (give each token a name, e.g. "ERP Production")
- See when a token was created and last used
- Revoke tokens at any time

How to use the token:

Include the token in every API request:

`Authorization: Bearer <your_api_token>`

Security recommendations:

- Treat the token like a password
- Store it in a secure secret store
- Rotate/revoke tokens regularly or when an integration changes.

3. Base URL, Versioning and Formats

Base URL:

- Production: `https://rrmprod.remitcloud.de/platform/api/v1/`
- Test: `https://rrmtest.remitcloud.de/platform/api/v1/`

Versioning:

All endpoints are prefixed with the version, e.g. `/platform/api/v1/...`.

Breaking changes will be introduced under a new version (e.g. `/platform/api/v2/...`).

Request / Response format:

Request body: JSON unless explicitly stated (e.g. file upload uses `multipart/form-data`)
Response: JSON
Character set: UTF-8

Standard response envelope:

```
{
  "data": { ... },
  "meta": { ... }
}
```

Field naming: All JSON fields use **snake_case** (e.g. `load_id`, `file_name`, `rrm_status`, `created_at`).

4. Identifiers

The API uses several identifiers. They are not interchangeable — each is assigned at a different stage, by a different party, and is used in different endpoints. This table is a map; detailed behaviour lives in the linked sections.

Identifier	Type	Assigned by	Where it's used	Summary
external_reference	UUID (RFC 4122)	Client	POST /reports, GET /reports?external_reference=	Your own idempotency key for JSON submissions. You generate it; unique per transaction within your account. Not used for XML upload. Submitting a duplicate returns 409 Conflict.
remit_file_id	integer	Platform	GET /reports/{remit_file_id}	Internal platform ID of a report transaction. Returned immediately by POST /reports, before any load_id exists. Always assigned on the JSON path. Not assigned for direct XML upload.
load_id	integer	Platform	section 8 ACER endpoints	Platform-generated ID for a file in the ACER processing flow. Assigned when the file enters processing — which is after JSON-schema and XSD validation, but happens regardless of the ACER business rules. A load_id can therefore exist for a file that was later rejected by those rules (RRMrejected) and never forwarded to ACER. Presence of a load_id means the file can be tracked — not that it was accepted or sent. The current acer_status (e.g. RRMaccepted / RRMrejected) reflects what actually happened, and it changes as the file progresses — poll it (or use webhooks) rather than treating the first value as final.
elcom_load_id	integer	Platform	section 9 ELCOM endpoints	ELCOM counterpart of load_id, with the same semantics: assigned when the file enters the ELCOM flow, present even if later rejected. Returned instead of / alongside load_id depending on whether

				your office has ACER, ELCOM, or both enabled (see section 9). Read the outcome from elcom_status.
webhook id	integer	Platform	DELETE /webhooks/{id}	Identifier of a registered webhook subscription.
unique_transaction_identifier (UTI)	string	Client	trade payload of POST /reports; inside uploaded XML	REMIT-level transaction identifier (Field 31) carried inside the report data. Identifies the trade to ACER. This is an in-payload field, not an API resource ID — there is no endpoint to look it up.

How the identifiers appear across the JSON flow

A JSON submission goes through two distinct stages, and which IDs exist tells you how far it got:

- **Submission** (`POST /reports`). The platform returns a `remit_file_id` immediately, then validates the payload against the JSON schema, generates the XML, and validates it against the XSD. The ACER business rules are not applied at this stage. If JSON-schema or XSD validation fails, the request is rejected and no `load_id` is produced — only `remit_file_id` exists.
- **Processing** (existing file flow). When the file enters processing it receives a `load_id` (and/or `elcom_load_id`). The ACER business rules are applied here. The file may pass and be forwarded, or be rejected (`RRMrejected`) — in which case the `load_id` still exists but the file is never sent onward.

So: a `remit_file_id` with no `load_id` means the submission did not pass schema/XSD validation (or has not yet been processed). A `load_id` that exists does not guarantee the file was accepted or sent — always read the live `acer_status` (or `elcom_status`) for the actual outcome. These statuses are dynamic and change as the file moves through the flow, so poll them or subscribe to webhooks instead of relying on a single read. Use the IDs to address and track a file; use the status fields to know what happened to it.

5. Status Codes and Error Handling

HTTP status codes:

Successful responses:

- `200` / `201` – Successful operation
- `204` – Successful operation with **no response body** (e.g. `DELETE /webhooks/{id}` — see section 10.2)

Client errors:

- `400` – **File rejected** after upload: the request was valid, but business or technical validation of the XML failed (`status: RRMrejected`). Response uses the standard `data` + `meta` envelope, **not** the `error` envelope below (see section 7.1)
- `401` – Authentication failed (missing/invalid token)
- `403` – Not allowed for this token

`404` – Resource not found (e.g. unknown `load_id`, `remit_file_id`, or webhook `id`)
`409` – Conflict (e.g. duplicate `external_reference` on `POST /reports`)
`422` – **Request validation error**: invalid or missing input fields. Response uses the `error` envelope with `code: VALIDATION_ERROR` and field-level `details` (e.g. `POST /reports`, `POST /webhooks`, invalid `POST /files` request such as missing file or wrong MIME type — see section 7.2)
`429` – Rate limit exceeded (see section 11)

Server errors:

`500` – Internal server error

Note: `400` and `422` both relate to failed input, but they use **different response shapes**. `400` on file upload means the file was received and processed, but rejected by RRM validation (`data` + `meta`). `422` means the HTTP request itself failed validation (`error` envelope).

Error body format (for endpoints that return `422` and other errors using the `error` envelope — **not** for `400` file rejection or `204 No Content`):

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data.",
    "details": {
      "field_name": ["Error description"]
    }
  }
}
```

6. Status Values

`The status` field can have one of the following values:

RRM statuses: `RRMrejected`, `RRMaccepted`
ACER statuses: `ACERrejected`, `ACERaccepted`, `ACERpartialAccepted`
System statuses: `SYSerror`, `SYSok`
Customer delivery: `SFTPtoCustomer`
ELCOM statuses: `ELCOMrejected`, `ELCOMaccepted`

7. Common Endpoints

7.1 Upload XML File

Purpose: Upload a new XML file for processing in RemitCloud. The file will be processed through ACER, ELCOM, or both, depending on your office configuration.
Method: `POST`

URL: `/platform/api/v1/files`

Authentication: Required (`Authorization: Bearer `)

Content-Type: `multipart/form-data`
Parameters:

Field	Required	Description
file	yes	XML file to upload, or a ZIP archive containing **exactly one XML file**

No other form fields are used. Schema type, reporting period, and business data are taken from the XML file name and content — not from separate request parameters.

Example request (multipart/form-data — not JSON):

```
POST /platform/api/v1/files HTTP/1.1
Authorization: Bearer <token>
Content-Type: multipart/form-data; boundary=----boundary

------boundary
Content-Disposition: form-data; name="file";
filename="20251126_REMITTable1_V3_file06.xml"
Content-Type: application/xml

<XML content>
------boundary--
```

File requirements:

Maximum upload size: 10 MB by default (server-configurable). Accepted formats: `.xml` (direct upload) or `.zip` (single XML inside).

File name must be valid (non-empty, max 255 characters)

Standard file naming (schema is detected from the file name):

```
[YYYYMMDD]_[SCHEMANAME]_[SCHEMAVERSION]_[TEXT].[EXTENSION]
```

Example: `20260623_REMITTable1_V1_TEXTContractTEXT.xml`

Segment	Description
YYYYMMDD	Submission date (ISO 8601): YYYY year, MM month, DD day
SCHEMANAME	Data schema name, e.g. REMITAnnexITable1
SCHEMAVERSION	Schema version, e.g. V1, V2
TEXT	Free text without spaces; allowed characters: a-z, A-Z, 0-9, _, -
EXTENSION	File extension: xml, csv, or bin (API upload accepts .xml directly or .zip containing a single XML file)

What happens after upload:

Schema is detected automatically (e.g. REMITTable1_V3, REMITTable2_V1) from the file name and XML content.

Technical and business validation runs.

On success, the file is stored and queued for ACER/ELCOM processing (per office settings).

Large files may be processed asynchronously; the response status is then `processing` — poll ACER/ELCOM endpoints using the returned `load_id`.

Note: `external_reference` belongs to section 7.2 (`POST /reports` JSON submissions) as an idempotency key. It is **not** used for XML file upload.

Response (accepted file, HTTP `201`):

```
{
```

```
  "data": {
    "load_id": 1345,
    "elcom_load_id": 789,
    "file_name": "20251126_REMITTable1_V3_file06.xml",
    "status": "RRMaccepted",
    "created_at": "2026-06-22T13:19:04+00:00"
  },
  "meta": {
    "schema_name": "REMITTable1_V3",
    "errors": {
      "technical": [],
      "business": []
    },
    "metadata": {}
  }
}
```

Response (rejected file, HTTP `400`):

```
{
  "data": {
    "load_id": 1345,
    "file_name": "20251126_REMITTable1_V3_file06.xml",
    "status": "RRMrejected",
    "created_at": "2026-06-22T13:19:04+00:00"
  },
  "meta": {
    "schema_name": "REMITTable1_V3",
    "errors": {
      "technical": ["Error description"],
      "business": []
    },
    "metadata": {}
  }
}
```

Notes:

- `load_id` is an integer that uniquely identifies the file for ACER processing (present if ACER is enabled for your office)
- `elcom_load_id` is an integer that uniquely identifies the file for ELCOM processing (present if ELCOM is enabled for your office)
- If your office has only ACER enabled, only `load_id` will be returned
- If your office has only ELCOM enabled, only `elcom_load_id` will be returned
- If your office has both ACER and ELCOM enabled, both `load_id` and `elcom_load_id` will be returned
- Use the appropriate ID (`load_id` for ACER endpoints, `elcom_load_id` for ELCOM endpoints) in subsequent API calls
- The `status` field is `RRMaccepted` or `RRMrejected` after initial RRM validation; rejected uploads may still include a `load_id` for tracking
- `meta.schema_name` contains the schema detected from your file.

7.2 Reports Transaction Endpoints

Submit REMIT Table 1 / Table 2 transactions as **JSON** instead of uploading XML.

All endpoints below are under `/platform/api/v1/reports` and require authentication.

Supported `schema_type` values:

Table	`schema_type`
REMIT Table 1 (REMITTable1_V3)	`remit_table_1`
REMIT Table 2 (REMITTable2_V1)	`remit_table_2`

The request body uses a nested **`trade`** object (not a flat `deal` object).
After a successful submission the platform generates REMIT XML, stores the trade JSON, and queues the file for ACER/ELCOM processing.

Endpoints in this section:

Method	URL	Purpose
POST	/platform/api/v1/reports	Submit a new transaction (JSON)
GET	/platform/api/v1/reports?external_reference={uuid}	Find Report Transaction by external_reference (Client UUID)
GET	/platform/api/v1/reports/{remit_file_id}	Get Report Transaction by Platform ID (remit_file_id)

`Example request` (`POST /reports`, Table 1 — abbreviated; full `trade` fields in section 14):

```
{
  "schema_type": "remit_table_1",
  "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "trade": {
    "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
    "other_market_participant": { "scheme": "ace", "value": "A00999002.NL" },
    "trading_capacity": "P",
    "buy_sell_indicator": "B",
    "contract_info": {
      "contract": {
        "contract_id": "NA",
        "contract_name": "EXECUTION",
        "contract_type": "FW",
        "energy_commodity": ["LG"],
        "settlement_method": "P",
        "organised_market_place": { "scheme": "bil", "value": "XBIL" },
        "delivery_point_or_zone": ["37Y005053MH0000R"],
        "delivery_start_date": "2026-04-06",
        "delivery_end_date": "2026-04-06",
        "duration": "N",
        "delivery_profiles": [{ "time_intervals": [{ "load_delivery_start_time":
"00:00:00", "load_delivery_end_time": "19:59:59" }] }]
      }
    },
    "organised_market_place": { "scheme": "bil", "value": "XBIL" },
    "transaction_time": "2026-05-06T17:51:17+02:00",
    "unique_transaction_identifier": { "value":
"6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001" },
    "price_details": { "price": 172.1888, "price_currency": "EUR" },
    "notional_amount_details": { "notional_amount": 10710249.6, "notional_currency":
```

```
"EUR" },
    "total_notional_contract_quantity": { "value": 62200, "unit": "KTherm" },
    "action_type": "N"
  }
}
```

Example response — immediately after `POST` (`201 Created`; pipeline IDs usually not yet available):

`remit_file_id` and `file_name` are assigned by the server; `created_at` is ISO-8601 with timezone offset (e.g. `+00:00`).

```
{
  "data": {
    "remit_file_id": 992,
    "created_at": "2026-06-22T13:19:04+00:00",
    "file_name": "20260622_REMITTable1_V3_NmEzOTM2_992.xml",
    "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "acer_status": "pending"
  },
  "meta": {
    "schema_type": "remit_table_1",
    "schema_name": "REMITTable1_V3"
  }
}
```

Example response — after polling `GET /reports/{remit_file_id}` (pipeline linked; `load_id` / `elcom_load_id` present):

```
{
  "data": {
    "remit_file_id": 992,
    "created_at": "2026-06-22T13:19:04+00:00",
    "file_name": "20260622_REMITTable1_V3_NmEzOTM2_992.xml",
    "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "acer_status": "ACERaccepted",
    "load_id": 1345,
    "elcom_load_id": 789,
    "elcom_status": "ELCOMaccepted",
    "record": {
      "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
      "transaction_time": "2026-05-06T17:51:17+02:00",
      "unique_transaction_identifier": { "value":
"6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001" },
      "action_type": "N"
    }
  },
  "meta": {
    "schema_type": "remit_table_1",
    "schema_name": "REMITTable1_V3"
  }
}
```

Example error response — duplicate `external_reference` (`409 Conflict`):

```
{
  "error": {
    "code": "CONFLICT_DUPLICATE_EXTERNAL_REFERENCE",
    "message": "A report with this external_reference already exists.",
    "details": {
      "external_reference": ["A report with this external_reference already exists."]
    }
  }
}
```

Example error response — validation failure (`422 Unprocessable Entity`):

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data.",
    "details": {
      "trade.market_participant.value": ["The trade.market participant.value field is
required."]
    }
  }
}
```

Tracking workflow:

- `POST /reports` returns **`remit_file_id`** and **`external_reference`** immediately (plus `acer_status`, typically `pending`).
- Poll **`GET /reports/{remit_file_id}`** or **`GET /reports?external_reference=…`** until `load_id` / `elcom_load_id` appear (when the file is linked in the ACER/ELCOM pipeline).
- Use those IDs with the ACER/ELCOM endpoints in sections 8–9 for detailed history, responses, and downloads.

`load_id` / `elcom_load_id` are **not guaranteed** in the `201 Created` response — they are added once backend processing creates the pipeline records.
See section 14 for the **complete JSON → TRUM field mapping** for Table 1 and Table 2.

7.2.1 Submit Report Transaction (POST)

Purpose: Create a new REMIT report transaction from JSON.
Method: `POST`

URL: `/platform/api/v1/reports`

Authentication: Required

Content-Type: `application/json`
Request body (top level):

Field	Required	Description
schema_type	yes	remit_table_1 or remit_table_2
external_reference	yes	Client UUID (RFC 4122); idempotency key
trade	yes	Nested transaction object (see section 14)

Example request – Table 1 (see 14.1 for all fields):

```
{
  "schema_type": "remit_table_1",
  "external_reference": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "trade": {
    "market_participant": {
      "scheme": "ace",
      "value": "A00999001.DE"
    },
    "other_market_participant": {
```

```
      "scheme": "ace",
      "value": "A00999002.NL"
    },
    "trading_capacity": "P",
    "buy_sell_indicator": "B",
    "contract_info": {
      "contract": {
        "contract_id": "NA",
        "contract_name": "EXECUTION",
        "contract_type": "FW",
        "energy_commodity": ["LG"],
        "settlement_method": "P",
        "organised_market_place": { "scheme": "bil", "value": "XBIL" },
        "delivery_point_or_zone": ["37Y005053MH0000R"],
        "delivery_start_date": "2026-04-06",
        "delivery_end_date": "2026-04-06",
        "duration": "N",
        "delivery_profiles": [{ "time_intervals": [{ "load_delivery_start_time":
"00:00:00", "load_delivery_end_time": "19:59:59" }] }]
      }
    },
    "organised_market_place": { "scheme": "bil", "value": "XBIL" },
    "transaction_time": "2026-05-06T17:51:17+02:00",
    "unique_transaction_identifier": { "value":
"6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001" },
    "price_details": { "price": 172.1888, "price_currency": "EUR" },
    "notional_amount_details": { "notional_amount": 10710249.6, "notional_currency":
"EUR" },
    "total_notional_contract_quantity": { "value": 62200, "unit": "KTherm" },
    "action_type": "N"
  }
}
```

Example request — Table 2 (see 14.2 for all fields):

```
{
  "schema_type": "remit_table_2",
  "external_reference": "b2c3d4e5-f6a7-8901-bcde-f12345678901",
  "trade": {
    "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
    "other_market_participants": [{ "scheme": "ace", "value": "A00999002.NL" }],
    "beneficiaries": [],
    "trading_capacity": "P",
    "buy_sell_indicator": "B",
    "contract_date": "2026-05-28",
    "contract_type": "FW",
    "energy_commodity": ["NG"],
    "price_or_price_formula": { "price": { "value": 334, "currency": "EUR" } },
    "total_notional_contract_quantity": { "value": 456, "unit": "KTherm" },
    "volume_optionality_intervals": [{
      "capacity": { "value": 50, "unit": "KTherm/d" },
      "start_date": "2025-12-03",
      "end_date": "2025-12-12"
    }],
    "settlement_method": "P",
    "delivery_point_or_zone": ["63W631527814486R"],
    "delivery_start_date": "2025-12-03",
    "delivery_end_date": "2025-12-05",
    "load_type": "BL",
    "action_type": "N"
  }
}
```

Example response (201 Created):

```
{
  "data": {
    "remit_file_id": 992,
    "created_at": "2026-06-22T13:19:04+00:00",
    "file_name": "20260622_REMITTable1_V3_NmEzOTM2_992.xml",
    "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "acer_status": "pending"
  },
  "meta": {
    "schema_type": "remit_table_1",
    "schema_name": "REMITTable1_V3"
  }
}
```

Notes:

- Save `remit_file_id` and/or `external_reference` — use them to poll 7.2.2 / 7.2.3 for updated status
- `load_id` and/or `elcom_load_id` may appear in later GET responses once ACER/ELCOM processing has linked the generated XML (depends on office pipeline configuration)
- Duplicate `external_reference` for the same user returns HTTP `409 Conflict` (see example above)
- Validation errors return HTTP `422` with field-level details (see example above).

7.2.2 Find Report Transaction by external_reference (Client UUID)

Purpose: Look up a previously submitted transaction by your `external_reference` (Client UUID).

Method: `GET`

URL: `/platform/api/v1/reports?external_reference={uuid}`

Authentication: Required

Query parameters:

Parameter	Required	Description
`external_reference`	yes	Client UUID you supplied when submitting the transaction

Example request:

```
GET /platform/api/v1/reports?external_reference=f47ac10b-58cc-4372-a567-0e02b2c3d479
HTTP/1.1
Authorization: Bearer <token>
```

Example response (200 OK):

```
{
  "data": {
    "remit_file_id": 992,
    "created_at": "2026-06-22T13:19:04+00:00",
    "file_name": "20260622_REMITTable1_V3_NmEzOTM2_992.xml",
    "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "acer_status": "ACERaccepted",
    "load_id": 1345,
    "elcom_load_id": 789,
```

```
    "elcom_status": "ELCOMaccepted",
    "record": {
      "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
      "transaction_time": "2026-05-06T17:51:17+02:00",
      "unique_transaction_identifier": { "value":
"6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001" },
      "action_type": "N"
    }
  },
  "meta": {
    "schema_type": "remit_table_1",
    "schema_name": "REMITTable1_V3"
  }
}
```

Example error response (`404 Not Found`):

```
{
  "error": {
    "code": "NOT_FOUND",
    "message": "Resource not found."
  }
}
```

Errors: `404` if no transaction with that reference exists for your office.

7.2.3 Get Report Transaction by Platform ID (remit_file_id)

Purpose: Retrieve full details of a report transaction by Platform ID (`remit_file_id`), including processing status and the stored trade record.

Method: `GET`

URL: `/platform/api/v1/reports/{remit_file_id}`

Authentication: Required

Path parameter:

`{remit_file_id}` (integer): Platform ID returned as `remit_file_id` from POST or GET lookup

Example request:

```
GET /platform/api/v1/reports/992 HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "remit_file_id": 992,
    "created_at": "2026-06-22T13:19:04+00:00",
    "file_name": "20260622_REMITTable1_V3_NmEzOTM2_992.xml",
    "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
    "acer_status": "ACERaccepted",
    "load_id": 1345,
    "elcom_load_id": 789,
    "elcom_status": "ELCOMaccepted",
    "record": {
      "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
      "transaction_time": "2026-05-06T17:51:17+02:00",
      "unique_transaction_identifier": { "value":
"6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001" },
      "action_type": "N"
```

```
    }
  },
  "meta": {
    "schema_type": "remit_table_1",
    "schema_name": "REMITTable1_V3"
  }
}
```

Notes:

- `record` contains the stored `trade` fields (null values are omitted)
- `acer_status` / `elcom_status` reflect current pipeline state; poll this endpoint until `load_id` / `elcom_load_id` are present, then use ACER/ELCOM endpoints (sections 8–9) for detailed history and receipt files.

8. ACER Endpoints

All ACER-related endpoints are under the `/platform/api/v1/acer/` prefix and use `load_id` to identify files.

For JSON report submissions, obtain `load_id` from `GET /reports/{remit_file_id}` (or by `external_reference`) once processing has linked the file — it is usually not available immediately after `POST /reports`.

8.1 List Files

Purpose: Retrieve a list of uploaded ACER files (for monitoring, reconciliation, etc.).
Method: `GET`

URL: `/platform/api/v1/acer/files`

Authentication: Required
Query parameters (all optional):

- `load_id` (integer): Filter by exact `load_id`
- `status` (string): Filter by status (e.g. `RRMaccepted`, `ACERaccepted`)
- `from` / `to` (ISO datetime): Filter by creation date range
- `page` (integer): Page number for pagination
- `per_page` (integer): Items per page (default: 50, max: 100)

Example request:

```
GET /platform/api/v1/acer/files?status=ACERaccepted&page=1&per_page=50 HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": [
    {
      "load_id": 1345,
      "file_name": "20251126_REMITTable1_V3_file06.xml",
      "status": "ACERaccepted",
      "created_at": "2026-06-22T13:19:04+00:00",
      "last_updated_at": "2026-06-22T13:23:04+00:00"
    }
  ],
  "meta": {
```

```
    "page": 1,
    "per_page": 50,
    "total": 1
  }
}
```

8.2 Get File by load_id

Purpose: Get the current status and key information for a specific ACER file.
Method: GET

URL: `/platform/api/v1/acer/files/{load_id}`

Authentication: Required
Path parameter:

`{load_id}` (integer): The ACER `load_id` (from file upload response, or from `GET /reports/…` once available)

Example request:

```
GET /platform/api/v1/acer/files/1345 HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "load_id": 1345,
    "file_name": "20251126_REMITTable1_V3_file06.xml",
    "rrm_status": "RRMaccepted",
    "acer_status": "ACERaccepted",
    "created_at": "2026-06-22T13:19:04+00:00",
    "last_updated_at": "2026-06-22T13:19:04+00:00",
    "raw_xml_download_url": "/platform/api/v1/acer/files/1345/download"
  }
}
```

8.3 Get File History

Purpose: Retrieve the processing history/events for an ACER file, including submissions and responses.
Method: GET

URL: `/platform/api/v1/acer/files/{load_id}/history`

Authentication: Required
Example request:

```
GET /platform/api/v1/acer/files/1345/history HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": [
    {
      "timestamp": "2026-06-22T13:19:04+00:00",
      "event": "UPLOADED",
      "status": "RRMaccepted",
      "details": {
        "rrm_receipt_file": "ReceiptRRM_20251126_REMITTable1_V3_file06.xml"
```

```
      }
    },
    {
      "timestamp": "2026-06-22T13:19:04+00:00",
      "event": "SENT_TO_ACER",
      "status": "SYSok",
      "details": {
        "sent_file": "20251126_REMITTable1_V3_B0001064H.DE_47.xml"
      }
    },
    {
      "timestamp": "2026-06-22T13:19:04+00:00",
      "event": "ACER_RESPONSE_RECEIVED",
      "status": "ACERaccepted",
      "details": {
        "acer_receipt_file": "ReceiptACER_20251126_REMITTable1_V3_file06.xml"
      }
    }
  ]
}
```

Notes:

- The history includes all processing steps and status changes
- Receipt filenames (RRM and ACER) are included in the `details` field when available
- Events may include: `UPLOADED`, `SENT_TO_ACER`, `ACER_RESPONSE_RECEIVED`, etc.

8.4 Download XML by load_id

Purpose: Download the originally uploaded XML file for a given `load_id`.
Method: `GET`

URL: `/platform/api/v1/acer/files/{load_id}/download`

Authentication: Required
Example request:

```
GET /platform/api/v1/acer/files/1345/download HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

- **Content-Type**: `application/xml`
- **Content-Disposition**: `attachment; filename=".xml"`

The response body contains the XML content (not JSON).

8.5 Get RRM Response

Purpose: Retrieve the RRM response for a given `load_id`.
Method: `GET`

URL: `/platform/api/v1/acer/files/{load_id}/rrm-response`

Authentication: Required
Path parameter:

- `{load_id}` (integer): The `load_id` of the file

Example request:

```
GET /platform/api/v1/acer/files/1345/rrm-response HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "load_id": 1345,
    "status": "RRMaccepted",
    "rrm_receipt_file": "ReceiptRRM_20251126_REMITTable1_V3_file06.xml",
    "raw_response": "<xml>...</xml>",
    "received_at": "2026-06-22T13:19:04+00:00"
  }
}
```

8.6 Get ACER Response

Purpose: Retrieve the ACER response for a given load_id.
Method: `GET`

URL: `/platform/api/v1/acer/files/{load_id}/response`

Authentication: Required
Path parameter:

`{load_id}` (integer): The `load_id` of the file

Example request:

```
GET /platform/api/v1/acer/files/1345/response HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "load_id": 1345,
    "status": "ACERaccepted",
    "acer_receipt_file": "ReceiptACER_20251126_REMITTable1_V3_file06.xml",
    "raw_response": "<xml>...</xml>",
    "received_at": "2026-06-22T13:19:04+00:00"
  }
}
```

9. ELCOM Endpoints

`All ELCOM-related endpoints are under the /platform/api/v1/elcom/` prefix and use `elcom_load_id` to identify files.

For JSON report submissions, obtain `elcom_load_id` from `GET /reports/{remit_file_id}` once ELCOM processing has linked the file.

9.1 List Files

Purpose: Retrieve a list of uploaded ELCOM files (for monitoring, reconciliation, etc.).
Method: `GET`

URL: `/platform/api/v1/elcom/files`

Authentication: Required
Query parameters (all optional):

- `elcom_load_id` (integer): Filter by exact `elcom_load_id`
- `status` (string): Filter by status (e.g. `ELCOMaccepted`, `ELCOMrejected`)
- `from` / `to` (ISO datetime): Filter by creation date range
- `page` (integer): Page number for pagination
- `per_page` (integer): Items per page (default: 50, max: 100)

Example request:

```
GET /platform/api/v1/elcom/files?status=ELCOMaccepted&page=1 HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": [
    {
      "elcom_load_id": 789,
      "file_name": "20251126_REMITTable1_V3_file06.xml",
      "status": "ELCOMaccepted",
      "created_at": "2026-06-22T13:19:04+00:00",
      "last_updated_at": "2026-06-22T13:19:04+00:00"
    }
  ],
  "meta": {
    "page": 1,
    "per_page": 50,
    "total": 1
  }
}
```

9.2 Get File by elcom_load_id

Purpose: Get the current status and key information for a specific ELCOM file.
Method: GET

URL: `/platform/api/v1/elcom/files/{elcom_load_id}`

Authentication: Required
Path parameter:

- `{elcom_load_id}` (integer): The ELCOM load ID (from file upload response, or from `GET /reports/…` once available)

Example request:

```
GET /platform/api/v1/elcom/files/789 HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "elcom_load_id": 789,
    "file_name": "20251126_REMITTable1_V3_file06.xml",
    "elcom_status": "ELCOMaccepted",
    "created_at": "2026-06-22T13:19:04+00:00",
    "last_updated_at": "2026-06-22T13:19:04+00:00",
    "raw_xml_download_url": "/platform/api/v1/elcom/files/789/download"
  }
```

```
}
```

9.3 Get File History

Purpose: Retrieve the processing history/events for an ELCOM file, including submissions and responses.
Method: GET

URL: `/platform/api/v1/elcom/files/{elcom_load_id}/history`

Authentication: Required
Example request:

```
GET /platform/api/v1/elcom/files/789/history HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": [
    {
      "timestamp": "2026-06-22T13:19:04+00:00",
      "event": "UPLOADED",
      "status": "SYSok",
      "details": null
    },
    {
      "timestamp": "2026-06-22T13:19:04+00:00",
      "event": "ELCOM_RESPONSE_RECEIVED",
      "status": "ELCOMaccepted",
      "details": {
        "elcom_receipt_file": "ReceiptELCOM_20251126_REMITTable1_V3_file06.xml"
      }
    }
  ]
}
```

Notes:

- The history includes all processing steps and status changes
- Receipt filenames (ELCOM) are included in the `details` field when available
- Events may include: `UPLOADED`, `ELCOM_RESPONSE_RECEIVED`, etc.

9.4 Download XML by elcom_load_id

Purpose: Download the originally uploaded XML file for a given `elcom_load_id`.
Method: `GET`

URL: `/platform/api/v1/elcom/files/{elcom_load_id}/download`

Authentication: Required
Example request:

```
GET /platform/api/v1/elcom/files/789/download HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

- **Content-Type**: `application/xml`
- **Content-Disposition**: `attachment; filename=".xml"`

The response body contains the XML content (not JSON).

9.5 Get ELCOM Response

Purpose: Retrieve the ELCOM response for a given `elcom_load_id`.
Method: `GET`

URL: `/platform/api/v1/elcom/files/{elcom_load_id}/response`

Authentication: Required
Path parameter:

`{elcom_load_id}` (integer): The `elcom_load_id` of the file

Example request:

```
GET /platform/api/v1/elcom/files/789/response HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": {
    "elcom_load_id": 789,
    "status": "ELCOMaccepted",
    "elcom_receipt_file": "ReceiptELCOM_20251126_REMITTable1_V3_file06.xml",
    "raw_response": "<xml>...</xml>",
    "received_at": "2026-06-22T13:19:04+00:00"
  }
}
```

10. Additional Endpoints

10.1 Health / Status Check

Purpose: Verify that the API is reachable and operational. Useful for monitoring and health checks.
Method: GET

URL: `/platform/api/v1/status`

Authentication: Not required
Example request:

```
GET /platform/api/v1/status HTTP/1.1
```

Example response (`200 OK`):

```
{
  "data": {
    "status": "OK",
    "time": "2026-06-22T13:19:04+00:00"
  }
}
```

10.2 Webhooks for Status Updates

Purpose: Register callback URLs to receive automatic status updates when file processing events occur (e.g. when ACER or ELCOM responses are received), eliminating the need for polling.

Register webhook

Method: `POST`

URL: `/platform/api/v1/webhooks`

Authentication: Required

Content-Type: `application/json`

Example request:

```
{
  "url": "https://your-system.com/api/callbacks",
  "events": ["ACER_RESPONSE_RECEIVED", "ELCOM_RESPONSE_RECEIVED"]
}
```

Example response (`201 Created`):

```
{
  "data": {
    "id": 1,
    "url": "https://your-system.com/api/callbacks",
    "events": ["ACER_RESPONSE_RECEIVED", "ELCOM_RESPONSE_RECEIVED"],
    "secret": "whsec_..."
  }
}
```

10.3 List webhooks

Method: `GET`

URL: `/platform/api/v1/webhooks`

Authentication: Required

Example request:

```
GET /platform/api/v1/webhooks HTTP/1.1
Authorization: Bearer <token>
```

Example response (`200 OK`):

```
{
  "data": [
    {
      "id": 1,
      "url": "https://your-system.com/api/callbacks",
      "events": ["ACER_RESPONSE_RECEIVED", "ELCOM_RESPONSE_RECEIVED"],
      "secret_masked": "whsec_ab...xyz1",
      "disabled_at": null,
      "created_at": "2026-06-22T13:19:04+00:00"
    }
  ],
  "meta": {
    "total": 1
  }
}
```

10.4 Remove webhook

Method: `DELETE`

URL: `/platform/api/v1/webhooks/{id}`

Authentication: Required
Example request:
```
DELETE /platform/api/v1/webhooks/1 HTTP/1.1
Authorization: Bearer <token>
```
Example response: `204 No Content` (empty body)
Note: Webhook configuration and available events will be provided during API setup.

11. Rate Limiting and Usage Limits

To ensure optimal performance and fair usage, the following limits apply to all API requests:

Rate Limits:

- API requests are subject to rate limiting per authentication token
- Rate limits are configured based on your subscription plan and usage requirements
- When rate limits are exceeded, you will receive an HTTP `429 Too Many Requests` response

Rate limit headers are included in all responses:

- `X-RateLimit-Limit`: Maximum number of requests allowed
- `X-RateLimit-Remaining`: Number of requests remaining in the current window
- `X-RateLimit-Reset`: Time when the rate limit resets (Unix timestamp)

File Upload Limits:

- Maximum upload size: 10 MB by default (server-configurable). Accepted formats: `.xml` (direct upload) or `.zip` (single XML inside).
- File validation occurs immediately upon upload

Data Retention:

- Uploaded files and processing responses are retained according to your account's retention policy
- Historical data access is available for the configured retention period
- Contact your account manager for specific retention details

Best Practices:

- Implement exponential backoff when receiving `429` responses
- Cache responses when appropriate to reduce API calls
- Use webhooks (when available) instead of polling for status updates
- Monitor rate limit headers to optimize your request patterns

Specific limits for your account will be provided during API setup and onboarding.

12. Example Usage

Upload a file (common endpoint - returns load_id and/or elcom_load_id)

```
curl -X POST https://rrmtest.remitcloud.de/platform/api/v1/files \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -F "file=@/path/to/20251126_REMITTable1_V3_file06.xml"
```

ZIP upload (archive must contain exactly one XML file):

```
curl -X POST https://rrmtest.remitcloud.de/platform/api/v1/files \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -F "file=@/path/to/report.zip"
```

Get ACER file status (using load_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/acer/files/1345 \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Get RRM response (using load_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/acer/files/1345/rrm-response
\
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Get ACER response (using load_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/acer/files/1345/response \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Get ELCOM file status (using elcom_load_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/elcom/files/789 \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Get ELCOM response (using elcom_load_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/elcom/files/789/response \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Submit a REMIT Table 1 report transaction (JSON)

```
curl -X POST https://rrmtest.remitcloud.de/platform/api/v1/reports \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "schema_type": "remit_table_1",
    "external_reference": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "trade": {
      "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
      "other_market_participant": { "scheme": "ace", "value": "A00999002.NL" },
      "trading_capacity": "P",
      "buy_sell_indicator": "B",
      "contract_info": {
        "contract": {
          "contract_id": "NA",
          "contract_name": "EXECUTION",
          "contract_type": "FW",
          "energy_commodity": ["LG"],
          "settlement_method": "P",
          "organised_market_place": { "scheme": "bil", "value": "XBIL" },
          "delivery_point_or_zone": ["37Y005053MH0000R"],
          "delivery_start_date": "2026-04-06",
          "delivery_end_date": "2026-04-06",
          "duration": "N",
          "delivery_profiles": [{ "time_intervals": [{ "load_delivery_start_time":
"00:00:00", "load_delivery_end_time": "19:59:59" }] }]
        }
```

```
      },
      "organised_market_place": { "scheme": "bil", "value": "XBIL" },
      "transaction_time": "2026-05-06T17:51:17+02:00",
      "total_notional_contract_quantity": { "value": 62200, "unit": "KTherm" },
      "action_type": "N"
    }
  }'
```

Get Report Transaction by Platform ID (remit_file_id)

```
curl -X GET https://rrmtest.remitcloud.de/platform/api/v1/reports/992 \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

Find Report Transaction by external_reference (Client UUID)

```
curl -X GET "https://rrmtest.remitcloud.de/platform/api/v1/reports?
external_reference=f47ac10b-58cc-4372-a567-0e02b2c3d479" \
  -H "Authorization: Bearer YOUR_API_TOKEN"
```

13. Support

For API support, please contact your RemitCloud account manager or technical support team.

Contact Information:

Website: remitcloud.de

Email: support@remitcloud.de.

13. JSON Field Reference (TRUM mapping)

`Client-facing POST /platform/api/v1/reports` payload for Table 1 and Table 2 uses the **nested `trade` JSON** validated against `resources/schemas/remit_table1.schema` and `resources/schemas/remit_table2.schema`.

Each row maps a JSON path to the corresponding **ACER REMIT TRUM** data field. Where the TRUM number is not known, the mapping says **Unknown TRUM field**.

Request envelope (both tables):

JSON field	Required	TRUM mapping
schema_type	yes	— (API metadata: remit_table_1 or remit_table_2)
external_reference	yes	— (client UUID idempotency key, not a TRUM field)
trade	yes	— (container for all TRUM-mapped transaction fields)

13.1 REMIT Table 1 — remit_table_1 (REMITTable1_V3)

Example request (minimal)

```
{
  "schema_type": "remit_table_1",
  "external_reference": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "trade": {
    "market_participant": {
      "scheme": "ace",
      "value": "A00999001.DE"
```

```
    },
    "other_market_participant": {
      "scheme": "ace",
      "value": "A00999002.NL"
    },
    "trading_capacity": "P",
    "buy_sell_indicator": "B",
    "contract_info": {
      "contract": {
        "contract_id": "NA",
        "contract_name": "EXECUTION",
        "contract_type": "FW",
        "energy_commodity": ["LG"],
        "settlement_method": "P",
        "organised_market_place": {
          "scheme": "bil",
          "value": "XBIL"
        },
        "contract_trading_hours": [
          {
            "start_time": "00:00:00",
            "end_time": "00:00:00"
          }
        ],
        "delivery_point_or_zone": [
          "37Y005053MH0000R"
        ],
        "delivery_start_date": "2026-04-06",
        "delivery_end_date": "2026-04-06",
        "duration": "N",
        "delivery_profiles": [
          {
            "time_intervals": [
              {
                "load_delivery_start_time": "00:00:00",
                "load_delivery_end_time": "19:59:59"
              }
            ]
          }
        ]
      }
    },
    "organised_market_place": {
      "scheme": "bil",
      "value": "XBIL"
    },
    "transaction_time": "2026-05-06T17:51:17+02:00",
    "unique_transaction_identifier": {
      "value": "6EWvJA3hhlTXfMK1eqoZUdb9HluO3Kl5SQOlDa99Ut001"
    },
    "price_details": {
      "price": 172.1888,
      "price_currency": "EUR"
    },
    "notional_amount_details": {
      "notional_amount": 10710249.6,
      "notional_currency": "EUR"
    },
    "quantity": {
      "value": 17,
      "unit": "KWh/h"
```

```
    },
    "total_notional_contract_quantity": {
      "value": 62200,
      "unit": "KTherm"
    },
    "action_type": "N"
  }
}
```

All JSON fields → TRUM (Table 1)

JSON field	TRUM mapping
external_reference	— (client idempotency key, not a TRUM field)
schema_type	— (API metadata, not a TRUM field)
trade.action_type	TRUM Data Field (58) — Action type
trade.aggressor	TRUM Data Field (12) — Initiator/Aggressor
trade.buy_sell_indicator	TRUM Data Field (11) — Buy/sell indicator
trade.execution_time	Unknown TRUM field (execution time; relates to order reporting fields 13–21 in TRUM)
trade.linked_order_ids[]	Unknown TRUM field (linked order identifiers; order fields 13–21 in TRUM)
trade.linked_transaction_ids[]	TRUM Data Field (32) — Linked transaction ID
trade.termination_date	TRUM Data Field (43) — Termination date
trade.trader_id	TRUM Data Field (3) — ID of the trader and/or market participant as identified by the organised market place
trade.trading_capacity	TRUM Data Field (10) — Trading capacity of the market participant in field 1
trade.transaction_time	TRUM Data Field (30) — Transaction timestamp
trade.unique_transaction_identifier	TRUM Data Field (31) — Unique transaction ID (UTI), string or object with value
trade.voice_brokered	TRUM Data Field (34) — Voice-brokered
trade.beneficiary.scheme	TRUM Data Field (9) — Type of code used in field 8
trade.beneficiary.value	TRUM Data Field (8) — Beneficiary ID
trade.contract_info.contract_id	— (contract reference by ID only; no separate TRUM number)
trade.fixing_indices[].index_name	TRUM Data Field (36) — Index value (index name part)
trade.market_particip	TRUM Data Field (2) — Type of code used in field 1

ant.scheme	
trade.market_participant.value	TRUM Data Field (1) – ID of the market participant or counterparty (reporting party)
trade.notional_amount_details.notional_amount	TRUM Data Field (38) – Notional amount
trade.notional_amount_details.notional_currency	TRUM Data Field (39) – Notional currency
trade.organised_market_place.scheme	TRUM Data Field (27) – Organised market place ID / OTC (identifier type)
trade.organised_market_place.value	TRUM Data Field (27) – Organised market place ID / OTC
trade.other_market_participant.scheme	TRUM Data Field (5) – Type of code used in field 4
trade.other_market_participant.value	TRUM Data Field (4) – ID of the other market participant or counterparty
trade.price_details.price	TRUM Data Field (35) – Price
trade.price_details.price_currency	TRUM Data Field (37) – Price currency
trade.price_interval_quantity_details[].days_of_week	TRUM Data Field (53) – Days of the week (within price interval)
trade.price_interval_quantity_details[].interval_end_date	Unknown TRUM field (price interval end date)
trade.price_interval_quantity_details[].interval_start_date	Unknown TRUM field (price interval start date)
trade.price_interval_quantity_details[].quantity	TRUM Data Field (55) – Delivery capacity
trade.price_interval_quantity_details[].unit	TRUM Data Field (56) – Quantity unit used in field 55
trade.quantity.unit	TRUM Data Field (42) – Quantity unit for field 40
trade.quantity.value	TRUM Data Field (40) – Quantity/Volume
trade.total_notional_contract_quantity.unit	TRUM Data Field (42) – Quantity unit for field 41
trade.total_notional_contract_quantity.value	TRUM Data Field (41) – Total notional contract quantity
trade.unique_transaction_identifier.additional_uti_info	Unknown TRUM field (additional UTI information)
trade.unique_transaction_identifier.value	TRUM Data Field (31) – Unique transaction ID (UTI)

trade.contract_info.contract.contract_id	— (contract identifier in XML contractInfo/contract; no separate TRUM number)
trade.contract_info.contract.contract_name	TRUM Data Field (22) — Contract name
trade.contract_info.contract.contract_type	TRUM Data Field (23) — Contract type
trade.contract_info.contract.delivery_end_date	TRUM Data Field (50) — Delivery end date
trade.contract_info.contract.delivery_point_or_zone[]	TRUM Data Field (48) — Delivery point or zone
trade.contract_info.contract.delivery_start_date	TRUM Data Field (49) — Delivery start date
trade.contract_info.contract.duration	TRUM Data Field (51) — Duration
trade.contract_info.contract.energy_commodity[]	TRUM Data Field (24) — Energy commodity
trade.contract_info.contract.index_names[]	TRUM Data Field (25) — Fixing index or reference price
trade.contract_info.contract.last_trading_date_time	Unknown TRUM field (last trading date/time; not numbered in Table 1 TRUM excerpt)
trade.contract_info.contract.load_type	TRUM Data Field (52) — Load type
trade.contract_info.contract.settlement_method	TRUM Data Field (26) — Settlement method
trade.fixing_indices[].index_curr_values[].index_currency	TRUM Data Field (36) — Index value (currency part)
trade.fixing_indices[].index_curr_values[].index_value	TRUM Data Field (36) — Index value
trade.price_interval_quantity_details[].price_time_interval_quantity.currency	TRUM Data Field (57) — Price/time interval quantity (price currency)
trade.price_interval_quantity_details[].price_time_interval_quantity.value	TRUM Data Field (57) — Price/time interval quantity (price value)
trade.price_interval_quantity_details[].time_intervals[].interval_end_time	TRUM Data Field (57) — Price/time interval quantity (interval end)
trade.price_interval_quantity_details[].time_intervals[].interv	TRUM Data Field (57) — Price/time interval quantity (interval start)

al_start_time	
trade.contract_info.contract.contract_trading_hours[].date	TRUM Data Field (28) – Contract trading hours (date)
trade.contract_info.contract.contract_trading_hours[].end_time	TRUM Data Field (28) – Contract trading hours (end time)
trade.contract_info.contract.contract_trading_hours[].start_time	TRUM Data Field (28) – Contract trading hours (start time)
trade.contract_info.contract.delivery_profiles[].days_of_week[]	TRUM Data Field (53) – Days of the week
trade.contract_info.contract.delivery_profiles[].load_delivery_end_date	TRUM Data Field (54) – Load delivery intervals (end date)
trade.contract_info.contract.delivery_profiles[].load_delivery_start_date	TRUM Data Field (54) – Load delivery intervals (start date)
trade.contract_info.contract.option_details.option_exercise_dates[]	Unknown TRUM field (option exercise dates)
trade.contract_info.contract.option_details.option_style	Unknown TRUM field (option style on standard contract)
trade.contract_info.contract.option_details.option_type	Unknown TRUM field (option type on standard contract)
trade.contract_info.contract.organised_market_place.scheme	TRUM Data Field (27) – Organised market place ID / OTC (identifier type)
trade.contract_info.contract.organised_market_place.value	TRUM Data Field (27) – Organised market place ID / OTC
trade.contract_info.contract.delivery_profiles[].time_intervals[].load_delivery_end_time	TRUM Data Field (54) – Load delivery intervals (end time)
trade.contract_info.contract.delivery_profiles[].time_intervals[].load_delivery_start_time	TRUM Data Field (54) – Load delivery intervals (start time)
trade.contract_info.contract.option_details.option_strike_price.currency	Unknown TRUM field (option strike price currency)

trade.contract_info.contract.option_details.option_strike_price.value	Unknown TRUM field (option strike price value)

Participant object pattern (market_participant, other_market_participant, beneficiary):

```
"market_participant": {
  "scheme": "ace",
  "value": "A00999001.DE"
}
```

Sub-field	TRUM mapping
scheme	Type-of-code TRUM field for the parent participant (fields 2, 5, or 9 depending on parent)
value	Participant ID TRUM field for the parent (fields 1, 4, or 8 depending on parent)

13.2 REMIT Table 2 — remit_table_2 (REMITTable2_V1)

Example request (minimal)

```
{
  "schema_type": "remit_table_2",
  "external_reference": "sample-redddffddd1mit-table2-001zxcv",
  "trade": {
    "market_participant": { "scheme": "ace", "value": "A00999001.DE" },
    "other_market_participants": [{ "scheme": "ace", "value": "A00999002.NL" }],
    "beneficiaries": [],
    "trading_capacity": "P",
    "buy_sell_indicator": "B",
    "contract_date": "2026-05-28",
    "contract_type": "FW",
    "energy_commodity": ["NG"],
    "price_or_price_formula": { "price": { "value": 334, "currency": "EUR" } },
    "total_notional_contract_quantity": { "value": 456, "unit": "KTherm" },
    "volume_optionality_intervals": [{
      "capacity": { "value": 50, "unit": "KTherm/d" },
      "start_date": "2025-12-03",
      "end_date": "2025-12-12"
    }],
    "settlement_method": "P",
    "delivery_point_or_zone": ["63W631527814486R"],
    "delivery_start_date": "2025-12-03",
    "delivery_end_date": "2025-12-05",
    "load_type": "BL",
    "action_type": "N"
  }
}
```

All JSON fields → TRUM (Table 2)

JSON field	TRUM mapping
external_reference	— (client idempotency key, not a TRUM field)

schema_type	— (API metadata, not a TRUM field)
trade.action_type	TRUM Data Field (45) — Action type
trade.buy_sell_indicator	TRUM Data Field (10) — Buy/sell indicator
trade.contract_date	TRUM Data Field (12) — Contract date
trade.contract_id	TRUM Data Field (11) — Contract ID
trade.contract_type	TRUM Data Field (13) — Contract type
trade.delivery_end_date	TRUM Data Field (43) — Delivery end date (delivery profile)
trade.delivery_point_or_zone[]	TRUM Data Field (41) — Delivery point or zone (delivery profile)
trade.delivery_start_date	TRUM Data Field (42) — Delivery start date (delivery profile)
trade.energy_commodity[]	TRUM Data Field (14) — Energy commodity
trade.load_type	TRUM Data Field (44) — Load type (delivery profile)
trade.settlement_method	TRUM Data Field (31) — Settlement method (Table 2)
trade.trading_capacity	TRUM Data Field (9) — Trading capacity
trade.type_of_index_price	TRUM Data Field (24) — Type of index price
trade.volume_optionality	TRUM Data Field (21) — Volume classification type (Volume optionality)
trade.volume_optionality_frequency	TRUM Data Field (22) — Volume optionality frequency
trade.beneficiaries[].scheme	TRUM Data Field (8) — Type of code used in field 7
trade.beneficiaries[].value	TRUM Data Field (7) — Beneficiary ID
trade.estimated_notional_amount.currency	TRUM Data Field (16) — Estimated notional amount (currency)
trade.estimated_notional_amount.value	TRUM Data Field (16) — Estimated notional amount
trade.fixing_index_details[].first_fixing_date	TRUM Data Field (28) — Fixing index details (first fixing date)
trade.fixing_index_details[].fixing_frequency	TRUM Data Field (30) — Fixing index details (fixing frequency)
trade.fixing_index_details[].fixing_index	TRUM Data Field (25) — Fixing index details (index name)
trade.fixing_index_details[].fixing_index_source	TRUM Data Field (27) — Fixing index details (index source)
trade.fixing_index_details[].fixing_index_	TRUM Data Field (26) — Fixing index details (index type)

type	
trade.fixing_index_details[].last_fixing_date	TRUM Data Field (29) — Fixing index details (last fixing date)
trade.market_participant.scheme	TRUM Data Field (2) — Type of code used in field 1
trade.market_participant.value	TRUM Data Field (1) — ID of the market participant or counterparty
trade.option_details[].option_exercise_frequency	TRUM Data Field (36) — Option details (exercise frequency)
trade.option_details[].option_first_exercise_date	TRUM Data Field (34) — Option details (first exercise date)
trade.option_details[].option_index_source	TRUM Data Field (39) — Option details (index source)
trade.option_details[].option_index_type	TRUM Data Field (38) — Option details (index type)
trade.option_details[].option_last_exercise_date	TRUM Data Field (35) — Option details (last exercise date)
trade.option_details[].option_strike_index	TRUM Data Field (37) — Option details (strike index)
trade.option_details[].option_style	TRUM Data Field (32) — Option details (option style)
trade.option_details[].option_type	TRUM Data Field (33) — Option details (option type)
trade.other_market_participants[].scheme	TRUM Data Field (4) — Type of code used in field 3
trade.other_market_participants[].value	TRUM Data Field (3) — ID of the other market participant or counterparty
trade.price_or_price_formula.price_formula	TRUM Data Field (15) — Price or price formula (formula text)
trade.total_notional_contract_quantity.unit	TRUM Data Field (18) — Total notional contract quantity (unit)
trade.total_notional_contract_quantity.value	TRUM Data Field (18) — Total notional contract quantity
trade.volume_optionality_intervals[].end_date	TRUM Data Field (23) — Volume optionality intervals (end date)
trade.volume_optionality_intervals[].start_date	TRUM Data Field (23) — Volume optionality intervals (start date)
trade.option_details[].option_strike_price.currency	TRUM Data Field (40) — Option details (strike price currency)
trade.option_details[	TRUM Data Field (40) — Option details (strike price value)

].option_strike_price.value	
trade.price_or_price_formula.price.currency	TRUM Data Field (15) — Price or price formula (price currency)
trade.price_or_price_formula.price.value	TRUM Data Field (15) — Price or price formula (price value)
trade.volume_optionality_intervals[].capacity.unit	TRUM Data Field (23) — Volume optionality intervals (capacity unit)
trade.volume_optionality_intervals[].capacity.value	TRUM Data Field (23) — Volume optionality intervals (capacity value)

Participant arrays (other_market_participants[], beneficiaries[]) use the same {scheme, value} object as Table 1; see TRUM fields (3)/(4) and (7)/(8) in the table above.