INSIDE INFORMATION

www.inside-information.de

Webware RRM Service

API Documentation

Version 1.0

Webware internet solutions GmbH

Teichstraße 14-16
34130 Kassel
Germany

Content

1. Overview

The **Inside Information Platform API** extends the existing web interface of the ACER REMIT Inside Information Platform (inside-information.de) with a **technical interface** for automated publication of inside information (UMMs) for:

- Electricity
- Gas
- Other inside information

Main capabilities (planned):

- **Create and publish UMMs** directly via API using ACER-compliant XML formats.
- **Submit corrections** of already published UMMs; only the **latest published message in a thread** can be corrected.
- **Keep the existing web interface** unchanged – users can still create and manage UMMs manually; the API is an additional channel.

Important design decisions:

- Via API **only "Save & Publish"** is allowed.
  - Draft-only messages ("Save" without publication) are **not** supported by the API to avoid inconsistent states.
- API expects **ACER XML** (REMIT UMM schemas for Electricity, Gas, Other).
- The **message ID threading logic** is fully respected:
  - Only the **latest published** message with a given message ID base (e.g. …6_002 after …6_001) can be corrected.

2. Authentication

Authentication method: Bearer token

Where to get the token:
In the Inside Information Platform **web interface** (inside-information.de), under:

- **Office Settings → API Tokens** (exact menu wording to be aligned with UI)

Capabilities in the web UI:

- Create named API tokens per office (e.g. “SCADA Integration”, “Scheduling System”).
- See when each token was created and last used.
- Revoke tokens at any time (immediately invalidates API access for that token).

How to use the token:
Include the token in every API request:

```
Authorization: Bearer <your_api_token>
```

Security recommendations:

- Treat tokens like passwords (store in a secure secret store).
- Use separate tokens per system / environment.
- Rotate tokens regularly or when an integration changes.
- Revoke tokens immediately if you suspect a leak.

3. Base URL, Versioning and Formats

Base URLs:

- **Production**: https://platform.inside-information.de/api/v1/
- **Test**: https://test.inside-information.de/api/v1/

Exact paths can be aligned with deployment; this documentation assumes the /api/v1/ prefix for all endpoints.

Versioning:

- All endpoints are prefixed with the version, e.g. /api/v1/....
- Future breaking changes will be introduced under a new version (e.g. /api/v2/...).

Request / Response formats:

- **Request body**:
  - For UMM creation/correction: application/xml (ACER REMIT UMM XML).
- **Response**: JSON, wrapped in a standard envelope:
  - Success:

    ```
    {
      "data": { },
      "meta": { }
    }
    ```
  - Error:

    ```
    {
      "error": {
        "code": "ERROR_CODE",
        "message": "Human-readable message.",
        "details": { }
      }
    }
    ```
- **Character set**: UTF-8
- **Field naming in JSON**: snake_case (e.g. message_id, thread_base, published_at).

4. Error Handling

HTTP status codes (planned):

- 200 – Successful operation (e.g. correction accepted).
- 201 – Resource created (e.g. new UMM published).
- 400 – Validation or input error (invalid XML, schema mismatch, business rule violation).
- 401 – Authentication failed (missing/invalid token).
- 403 – Not allowed for this token / office.
- 404 – Resource not found (e.g. message_id unknown in this office).

- 409 – Conflict (e.g. attempt to correct a non-latest message in thread).
- 500 – Internal server error.

Error codes (values of error.code):

- AUTH_FAILED – authentication failed (missing/invalid token).
- FORBIDDEN – token is valid but not allowed to access the requested resource.
- VALIDATION_ERROR – request body or XML is invalid (schema or business rule violation).
- XML_INVALID – XML is not well-formed or does not match the ACER schema.
- NOT_FOUND – requested resource does not exist (e.g. message_id or UMM not found).
- CONFLICT_NOT_LATEST_IN_THREAD – correction attempted on a message that is not the latest in its thread.
- INTERNAL_ERROR – unexpected server-side error.

Error body format (HTTP status code + code + human-readable message + optional details):

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data.",
    "details": {
      "xml": ["XML does not match REMIT UMM Electricity schema."],
      "business_rule": ["Only latest published message in thread can be corrected."]
    }
  }
}
```

5. Message and Status Model

5.1 Message ID and Threading

Each UMM is identified in ACER XML by a **messageId**
(e.g. 0000000000000000000000006_001):

- **Thread base**: the part **before** the underscore,
  e.g. 0000000000000000000000006.

- **Sequence**: the part **after** the underscore, e.g. _001, _002, …

Threading rules:

- The **first** published message in a thread uses sequence _001.

- Every **correction** creates a **new message** in the same thread:
  - Sequence is incremented (_002, _003, …).

  - The previous message in the thread is marked as **old** and **dismissed** internally.

- At any point in time, **only the latest published message in a thread** is considered the active one.

Internally, the platform keeps additional technical information to manage the thread (for example, which message is the original and which messages are corrections), but these details are handled automatically by the system and are not relevant for API integrations.

5.2 Behaviour of API Requests

- API create and correct endpoints always attempt to **validate and publish** the UMM in a single step.

- If validation passes, the message is stored and appears in the Atom feed; if validation fails, the request is rejected with an appropriate HTTP error code and explanation in the response body.

6. UMM Endpoints

All UMM endpoints are under the /platform/api/v1/umm/ prefix and require authentication.

6.1 Create Electricity UMM

Purpose: Create and publish a new **Electricity** UMM from ACER-compliant XML.

Method: POST
URL: /api/v1/umm/electricity
Authentication: Required (Authorization: Bearer <token>)
Content-Type: application/xml

Request body:

- XML document compliant with ACER Electricity UMM schema, for example REMITUMMElectricitySchema_V2.xsd (schema version to be confirmed).
- messageId in the XML can be:
  - **Omitted** – server generates a proper messageId (<thread_base>_001) based on internal ID, or
  - Provided as placeholder – server overrides with canonical format.

Validation steps:

1. Token / office validation.
2. XML well-formedness.
3. XSD validation against the configured Electricity UMM schema.
4. Business rules (required fields, allowed event types, fuel type rules, etc.).

Behaviour:

- Message is **created and published** in a single step:
  - Internal status set to “published”.
  - Publication timestamp set to current time.
  - message_id generated in the format 0000000000000000000000<id>_001 (exact padding aligned with current system).
  - History record is written.

    - Optional email notification is sent based on office/user settings.

- No draft-only save via API; requests are either rejected or published.

Successful response (201 Created):

```
{
  "data": {
    "message_id": "0000000000000000000000123_001",
    "thread_base": "0000000000000000000000123",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T10:00:00Z",
    "event_status": "Active",
    "event_type": "Production unavailability"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMElectricitySchema_V2"
  }
}
```

6.2 Correct Electricity UMM

Purpose: Submit a **correction** to an already published Electricity UMM. Only the **latest published message in the thread** identified by the given message_id can be corrected.

In a typical integration flow, the client first **lists or fetches existing UMMs** (see section 7) to determine:

- Which UMMs exist for a given time period or asset, and

- Which exact message_id (and thread) should be corrected.

Once the client has identified the correct message_id, this correction endpoint can be called with the new XML version of the message.

Method: POST
URL: /api/v1/umm/electricity/{message_id}/correct
Authentication: Required
Content-Type: application/xml

Path parameter:

- {message_id} (string): Full messageId of any message in the thread (e.g. 0000000000000000000000006_001).

Request body:

- XML document compliant with ACER Electricity UMM schema.
- XML represents the **new complete state** of the message (not a diff).
- messageId in XML can:
  - Match {message_id}, or
  - Already be incremented (…_002); the server validates and enforces the correct next sequence.

Server logic (thread enforcement):

1. Extract **thread base** from {message_id} (part before underscore).
2. Find the **latest published, not-old** Electricity UMM in this thread for the current office.
3. If no such message exists → 404 NOT_FOUND.
4. If the found latest message's message_id does not equal {message_id} → 409 CONFLICT with code CONFLICT_NOT_LATEST_IN_THREAD.
5. If OK:
   - Mark the existing message as no longer current (internally flagged as an old entry).
   - Create a new message:
     - Map fields from XML.
     - Link it as the next message in the same message ID thread.
     - Compute new message_id by incrementing the sequence (_002, _003, …).
     - Set status "published" and publication timestamp to current time.
   - Write history and optionally send email.

Successful response (201 Created):

```
{
  "data": {
    "message_id": "0000000000000000000000123_002",
    "thread_base": "0000000000000000000000123",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T11:05:00Z",
    "previous_message_id": "0000000000000000000000123_001"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMElectricitySchema_V2"
  }
}
```

6.3 Create Gas UMM

Purpose: Create and publish a new **Gas** UMM from ACER-compliant XML.

Method: POST
URL: /api/v1/umm/gas
Authentication: Required
Content-Type: application/xml

Request body:

- XML document compliant with ACER Gas UMM schema (e.g. REMITUMMGasSchema_V2.xsd).
- messageId behaviour is the same as for Electricity:
    - Can be omitted → server generates <thread_base>_001.
    - If present, the server may override with canonical format.

Behaviour:

- Equivalent to the Electricity UMM creation, but using the Gas UMM schema and Gas-specific fields (balancing zones, technical capacity, direction, etc.).
- Message is validated and, if valid, stored and published in one step; no draft-only saves.

Successful response example:

```
{
  "data": {
    "message_id": "000000000000000000000055_001",
    "thread_base": "000000000000000000000055",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T10:15:00Z"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMGasSchema_V2"
  }
}
```

6.4 Correct Gas UMM

Purpose: Submit a **correction** to an already published Gas UMM, following the same threading rules as Electricity.

In a typical integration flow, the client first **lists or fetches existing Gas UMMs** (see section 7) to determine which message_id and thread should be corrected.

Method: POST
URL: /api/v1/umm/gas/{message_id}/correct
Authentication: Required
Content-Type: application/xml

Path parameter:

- {message_id} (string): Full messageId of any Gas UMM in the thread.

Request body:

- Gas UMM XML (REMIT Gas schema), representing the new complete state of the message.

Behaviour:

- Thread resolution and "only latest message can be corrected" work exactly like in Electricity:
    - If {message_id} is not the latest in its thread, the request is rejected with HTTP 409 and an explanatory error.
    - If it is the latest, a new Gas UMM is created with an incremented message_id (e.g. _002), and the previous one is marked as an old entry internally.

Successful response example:

```
{
  "data": {
    "message_id": "000000000000000000000055_002",
    "thread_base": "000000000000000000000055",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T11:15:00Z",
    "previous_message_id": "000000000000000000000055_001"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMGasSchema_V2"
  }
}
```

6.5 Create Other UMM

Purpose: Create and publish a new **Other** inside information UMM from ACER-compliant XML.

Method: POST
URL: /api/v1/umm/other
Authentication: Required
Content-Type: application/xml

Request body:

- XML document compliant with ACER "Other" UMM schema (e.g. REMITUMMOtherSchema_V1.xsd).

Behaviour:

- Same pattern as Electricity/Gas:
  - Validate XML and business rules.
  - On success, store and publish in one step.
  - Generate a new message_id thread starting with _001.

Successful response example:

```
{
  "data": {
    "message_id": "000000000000000000000099_001",
    "thread_base": "000000000000000000000099",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T10:30:00Z"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMOtherSchema_V1"
  }
}
```

6.6 Correct Other UMM

Purpose: Submit a **correction** to an already published “Other” UMM.

Method: POST
URL: /api/v1/umm/other/{message_id}/correct
Authentication: Required
Content-Type: application/xml

Path parameter:

- {message_id} (string): Full messageId of any “Other” UMM in the thread.

Request body:

- “Other” UMM XML (REMIT Other schema), representing the new complete state of the message.

Behaviour:

- Same correction and thread rules as Electricity and Gas:
  - Only the latest published message in the thread can be corrected.
  - A correction creates a new entry with incremented message_id; the previous one becomes an old entry internally.

Successful response example:

```
{
  "data": {
    "message_id": "0000000000000000000000099_002",
    "thread_base": "0000000000000000000000099",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T11:30:00Z",
    "previous_message_id": "0000000000000000000000099_001"
  },
  "meta": {
    "environment": "prod",
    "schema_version": "REMITUMMOtherSchema_V1"
  }
}
```

7. Read / Monitoring Endpoints (optional)

These endpoints are optional and can be implemented if needed by customers.

7.1 List UMMs for a Commodity

Purpose: List UMMs for a given commodity (electricity, gas, other), typically to find the correct message_id and thread before sending a correction.

Method: GET
URL: /api/v1/umm/{commodity}
{commodity} ∈ electricity | gas | other

Query parameters (all optional):

- message_id (string): Filter by exact message_id.
- thread_base (string): Filter by a specific thread base (part before _001, _002, …).
- from / to (ISO datetime): Filter by publication date/time range.
- status (string): Filter by status (e.g. PUBLISHED).
- page (integer): Page number for pagination (default: 1).
- per_page (integer): Items per page (default: 50, max: 100).

Response example:

```
{
 "data": [
  {
   "message_id": "0000000000000000000000123_001",
   "thread_base": "0000000000000000000000123",
   "status": "PUBLISHED",
   "published_at": "2025-11-27T10:00:00Z"
  },
  {
   "message_id": "0000000000000000000000123_002",
   "thread_base": "0000000000000000000000123",
   "status": "PUBLISHED",
   "published_at": "2025-11-27T11:05:00Z"
  }
  ],
  "meta": {
    "page": 1,
    "per_page": 50,
    "total": 2
  }
}
```

7.2 Get UMM by message_id

Method: GET
URL: /api/v1/umm/{commodity}/{message_id}
{commodity} ∊ electricity | gas | other

Response example:

```
{
  "data": {
    "message_id": "0000000000000000000000123_002",
    "thread_base": "0000000000000000000000123",
    "status": "PUBLISHED",
    "event_status": "Active",
    "event_type": "Production unavailability",
    "published_at": "2025-11-27T11:05:00Z",
    "xml_download_url":
"/api/v1/umm/electricity/0000000000000000000000123_002/download"
  }
}
```

7.3 Download UMM XML by message_id

Method: GET
URL: /api/v1/umm/{commodity}/{message_id}/download

Response:

- Content-Type: application/xml
- Body: stored UMM XML for that message.

8. Atom Feed and Publication Behaviour

The public Atom feed remains the **primary publication channel** for inside information.

- Only **published** UMMs (status = “published”) appear in the Atom feeds.

- API-created UMMs behave exactly like web-created UMMs:
    - Once accepted and published, they appear in the Atom feed.
    - Corrections create new Atom entries; older entries are marked internally as dismissed/old but remain in history.

This ensures that automation via API and manual operation in the web interface are **fully consistent**.

9. Rate Limiting and Usage

To ensure stability and fair usage, the following concepts apply (limits configurable per customer):

- **Rate limiting per API token**:
    - Configurable maximum number of requests per minute.
    - When exceeded, HTTP 429 Too Many Requests is returned.
    - Response headers:
        - X-RateLimit-Limit
        - X-RateLimit-Remaining
        - X-RateLimit-Reset (Unix timestamp).
- **UMM submission limits**:
    - XML size limited to a reasonable maximum.
    - Only ACER XML is accepted; no JSON representations for UMM content.

Best practices:

- Implement exponential backoff on 429 and 5xx responses.
- Avoid re-submitting the same XML repeatedly.

10. Example Usage

10.1 Create Electricity UMM (Test Environment)

```
curl -X POST "https://test.inside-information.de/api/v1/umm/electricity" \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -H "Content-Type: application/xml" \
  --data-binary @/path/to/your_electricity_umm.xml
```

Example JSON response:

```
{
  "data": {
    "message_id": "0000000000000000000000123_001",
    "thread_base": "0000000000000000000000123",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T10:00:00Z"
  }
}
```

10.2 Correct Electricity UMM

```
curl -X POST \
  "https://test.inside-
information.de/api/v1/umm/electricity/0000000000000000000000123_001/correct
" \
  -H "Authorization: Bearer YOUR_API_TOKEN" \
  -H "Content-Type: application/xml" \
  --data-binary @/path/to/your_corrected_electricity_umm.xml
```

Example JSON response on **successful** correction:

```
{
  "data": {
    "message_id": "0000000000000000000000123_002",
    "thread_base": "0000000000000000000000123",
    "status": "PUBLISHED",
    "published_at": "2025-11-27T11:05:00Z",
    "previous_message_id": "0000000000000000000000123_001"
  }
}
```

If 0000000000000000000000123_001 is **not** the latest in its thread (for example, ...123_002 already exists), the API responds:

```
{
  "error": {
    "code": "CONFLICT_NOT_LATEST_IN_THREAD",
    "message": "Only the latest published message in this thread can be corrected.",
    "details": {
      "latest_message_id": "0000000000000000000000123_002"
    }
  }
}
```

11. Support

For API feature availability, onboarding and technical support:

- Website: https://inside-information.de/
- Email: support@inside-information.de
- Phone: as published in the platform’s contact section.

Feature availability:

- This documentation describes an **additional API option** for the Inside Information Platform.
- The endpoints become available to offices only after the API feature is activated for the respective account.